FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Pattern-based Testing Framework for IoT Ecosystems

Pedro Martins Pontes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Pascoal Faria

Co-Supervisor: Bruno Lima

July 13, 2018

A Pattern-based Testing Framework for IoT Ecosystems

Pedro Martins Pontes

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Ana Paiva
External Examiner: Doctor Rui Maranhão
Supervisor: Doctor João Pascoal Faria
July 13, 2018

Abstract

The emergence of Internet of Things (IoT) technology is expected to bring forward new promising solutions in various domains and, consequently, impact many aspects of everyday life. However, the development and testing of software applications and services for IoT systems encompasses several challenges that existing solutions have not yet properly addressed.

Implementing test cases covering multiple aspects, interfaces, and protocols is a demanding task due to the heterogeneous and distributed nature of IoT systems. This fact — along with and the importance of testing in the development process — gives rise to the need for an efficient way to implement automated testing in IoT. Although there are already several tools that can be used in the testing of IoT systems, a number of issues can be pointed out: a technological review of existing solutions reveals the lack of a comprehensive test solution for automated integration testing. Focusing on a specific platform, language, or standard, limiting the possibility of improvement or extension, and not providing out-of-the-box functionality are among the most common shortcomings detected.

A pattern-based approach to testing IoT systems that aims to address these faults is proposed. As part of this approach, a feature model was devised, enabling the representation of the plurality of components and features of an IoT ecosystem. This was a first step in identifying a set of recurring behaviors of IoT applications and a set of corresponding test strategies, which are defined as test patterns specific to the IoT domain. A pattern-based test automation framework for integration testing of IoT ecosystems — *Izinto* — that implements in a generic way this set of test patterns which can be easily instantiated for concrete IoT scenarios is described.

Izinto was validated in a number of test cases, within a concrete application scenario in the domain of Ambient Assisted Living (AAL), which allowed to confirm its potential to reduce the effort that needs to be put into the test process. Furthermore, the framework enabled the identification of errors introduced both on purpose and by accident, proving its effectiveness.

Finally, some courses of investigation are laid out as future work, which can include the definition of additional IoT Test Patterns and further developing *Izinto*.

Resumo

A ascenção da Internet das Coisas — Internet of Things (IoT) — deverá trazer novas soluções promissoras em vários domínios e, consequentemente, ter impacto em muitos aspetos da vida cotidiana. No entanto, o desenvolvimento e o teste de aplicações e serviços de *software* para sistemas *IoT* acarretam vários desafios aos quais as soluções existentes ainda não dão uma resposta adequada.

A implementação de casos de teste que abrangem vários aspetos, interfaces e protocolos é uma tarefa exigente devido à natureza heterogénea e distribuída dos sistemas de *IoT*. Esse facto — juntamente com a importância que os testes assumem no processo de desenvolvimento — dá origem à necessidade de uma maneira eficiente de implementar testes automatizados em *IoT*. Embora já existam várias ferramentas que podem ser usadas no teste de sistemas *IoT*, podem ser-lhes apontadas várias lacunas: uma revisão tecnológica das soluções existentes revela a necessidade de encontrar uma solução de teste abrangente para testes de integração automatizados. Entre as limitações mais comuns encontram-se o focar numa única plataforma ou linguagem de programação, o limitar a possibilidade de melhoria ou extensão, e não fornecer funcionalidades prontas para uso.

É apresentada uma abordagem baseada em padrões para testar sistemas de *IoT* que visa colmatar essas falhas. Como parte dessa abordagem, um modelo de funcionalidades foi criado, permitindo a representação da pluralidade de componentes e funcionalidades de um ecossistema de *IoT*. Este foi o primeiro passo para identificar um conjunto de comportamentos recorrentes em aplicações *IoT* e um conjunto de estratégias de teste correspondentes, que são definidos como padrões de teste específicos para o domínio *IoT*. *Izinto* é uma *framework* de automação de testes baseada em padrões para testes de integração de ecossistemas *IoT* que implementa de forma genérica este conjunto de padrões de teste, que podem ser facilmente instanciados para cenários de *IoT* concretos.

Izinto foi validada em vários casos de teste, dentro de um cenário concreto de aplicação na área de *Ambient Assisted Living* (*AAL*), o que permitiu confirmar seu potencial para reduzir o esforço que precisa ser colocado no processo de teste. Além disso, a estrutura permitiu a identificação de erros introduzidos de propósito e por acidente, comprovando sua eficácia.

São ainda apresentadas algumas linhas de investigação para o trabalho futuro, que pode incluir a definição de outros padrões de teste *IoT* e o desenvolvimento da *framework*.

Acknowledgements

This section is reserved for the acknowledgement of all those who — directly or indirectly — played a part in the development of this master's thesis. Therefore, this a message of thanking and recognition for those people.

I would first like to thank my thesis supervisor, Professor João Pascoal Faria for his support, guidance, and oversight along the writing of this dissertation.

I would also like to acknowledge Bruno Lima as the co-supervisor of this dissertation work, and I am indebted to him for his invaluable comments and suggestions.

Additionally, I would like to thank all the professors, colleagues, and friends of the Faculty of Engineering of the University of Porto, for everything they taught me and for every memorable moment we shared these last five years.

Finally, I cannot pass up the opportunity to thank my family — my mother and my sister in particular —, for their unwavering support and continuous encouragement throughout my years of study and through the process of researching and writing this dissertation. This accomplishment would not have been possible without them.

Thank you.

Pedro Martins Pontes

"Science can amuse us all, but it is engineering that changes the world."

Isaac Asimov

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Objectives and Motivation . . . 4
- 1.3 Document Structure . . . 4

2 Background and State-of-the-Art **7**
- 2.1 IoT Concepts, Architecture and Technologies . . . 8
- 2.2 IoT for Healthcare Ecosystems . . . 10
- 2.3 Software Development for IoT . . . 13
  - 2.3.1 Challenges . . . 13
  - 2.3.2 Existing Solutions . . . 14
- 2.4 Testing for IoT . . . 16
  - 2.4.1 Challenges . . . 17
  - 2.4.2 Existing Solutions . . . 18
- 2.5 Patterns and IoT . . . 22
- 2.6 Summary . . . 23

3 A Feature Model and Test Patterns for IoT **25**
- 3.1 Feature Model . . . 26
- 3.2 Test Patterns . . . 28
  - 3.2.1 Test Periodic Readings . . . 29
  - 3.2.2 Test Triggered Readings . . . 31
  - 3.2.3 Test Alerts . . . 33
  - 3.2.4 Test Actions . . . 35
  - 3.2.5 Test Actuators . . . 37
- 3.3 Summary . . . 38

4 *Izinto* Test Framework **39**
- 4.1 Functionality . . . 39
- 4.2 Architecture and Implementation . . . 41
  - 4.2.1 Test Logic Module . . . 42
  - 4.2.2 IoT Module . . . 45
- 4.3 Usage . . . 48
  - 4.3.1 Test Duration . . . 49
  - 4.3.2 Data Sources . . . 49
  - 4.3.3 Devices . . . 50
  - 4.3.4 Triggers . . . 51
- 4.4 Summary . . . 53

5 Validation **55**
5.1 Application Scenario . . . 56
5.1.1 Functionality . . . 57
5.1.2 Components . . . 59
5.1.3 Architecture and Implementation . . . 62
5.2 Test cases . . . 67
5.2.1 Temperature/Humidity Test . . . 70
5.2.2 Heart Rate Monitoring Test . . . 71
5.2.3 Air Quality Monitoring Test . . . 72
5.2.4 Luminosity Test . . . 73
5.2.5 Weight Monitoring Test . . . 74
5.2.6 Blood Pressure Monitoring Test . . . 75
5.3 Results . . . 76
5.4 Discussion . . . 77
5.5 Summary . . . 78

6 Conclusion **79**
6.1 Overview . . . 79
6.2 Contributions . . . 80
6.3 Future Work . . . 81
6.3.1 IoT Test Patterns . . . 81
6.3.2 IoT Testing Framework . . . 81

References **83**

A Configuration File Example **93**

B Scientific Publications **103**
B.1 *Izinto*: A Pattern-based IoT Testing Framework . . . 103
B.2 Test Patterns for IoT . . . 110

List of Figures

1.1 Evolution in the number of IoT connected devices worldwide from 2015 to 2025. 2
1.2 Population aged 60 or over: world and development regions, 1950-2050. 3

2.1 Three-layered architecture. 9
2.2 Detailed view of the emerging end-to-end IoT architecture, including its individual elements. 9
2.3 Edge-Fog Cloud architecture for IoT. 10
2.4 Levels of testing across IoT layers. 16

3.1 Feature model for an IoT ecosystem. 27

4.1 Representation of the framework's architecture. 41
4.2 Package *TestHandlers*, corresponding to the module encompassing test logic. . . 42
4.3 Order of execution of methods with different JUnit annotations. 43
4.4 Relationship of dependency of *DataSources* on *Readings*. 46
4.5 Relation between the classes related to devices and their configuration. 47
4.6 Relation between the classes related to triggers, conditions, and outcomes. 47

5.1 Architecture of the application scenario. 56
5.2 Main components used in the application scenario. 59
5.3 Deployment view depicting the connection between the body sensors and the system's gateway. 63
5.4 Assembling of the ambient sensors and the ESP32 module. 64
5.5 Deployment view depicting the connection between the ambient sensors and the system's gateway. 64
5.6 Deployment view depicting the connection between the actuators and the system's gateway. 65
5.7 Dashboard's widgets for the visualization of real time data. 65
5.8 Dashboard's widgets the visualization of historic data. 66
5.9 Dashboard's interface for alert management. 66
5.10 Deployment view depicting the connection between the system's gateway and the system's server. 67
5.11 Architecture illustrating the test case for temperature/humidity features. 70
5.12 Architecture illustrating the test case for heart rate monitoring features. 71
5.13 Architecture illustrating the test case for air quality monitoring features. 72
5.14 Architecture illustrating the test case for luminosity features. 73
5.15 Architecture illustrating the test case for weight monitoring features. 74
5.16 Architecture illustrating the test case for blood pressure monitoring features. . . 75
5.17 Example of output for a successful test run. 76

5.18 Example of output for an unsuccessful test run. 77

List of Tables

2.1 Related projects in the area of remote health monitoring: summary of deployment scenarios and technologies. . . . 11
2.2 Summary comparison of available IoT testing tools. . . . 21

3.1 Test patterns identified. . . . 28

4.1 Attributes for the class *Reading*. . . . 46

5.1 Categorization of heart rate for different intervals, defined as a percentage of the MHR. . . . 57
5.2 Healthy and unhealthy blood pressure ranges, as defined by the AHA. . . . 57
5.3 Weight thresholds considered for the triggering of alerts. . . . 58
5.4 Ambient condition parameters considered in the scenario and corresponding thresholds for the triggering of alerts. . . . 58
5.5 Triggering conditions and respective actions considered in the scenario. . . . 58
5.6 Devices and corresponding features. . . . 68
5.7 Outcome of different test cases considered. . . . 76

Abbreviations

AAL	Ambient Assisted Living
AHA	American Heart Association
AC	Air Conditioner
API	Application Programming Interface
BLE	Bluetooth Low Energy
BPM	Beats Per Minute
CI	Continuous Integration
CoAP	Constrained Application Protocol
EHR	Electronic Health Record
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
ISSTA	International Symposium on Software Testing and Analysis
IT	Integration Testing
LPWAN	Low Power Wireless Personal Area Network
MBT	Model-based Testing
MHR	Maximum Heart Rate
MQTT	Message Queue Telemetry Transport
NFC	Near Field Communication
PPM	Parts Per Million
REST	Representational State Transfer
RFID	Radio Frequency Identification
RPM	Remote Patient Monitoring
ST	System Testing
SUT	System Under Test
UI	User Interface
UML	Unified Modelling Language
UT	Unit Testing
WBSN	Wireless Body Sensor Network

Chapter 1

Introduction

1.1 Context 1
1.2 Objectives and Motivation 4
1.3 Document Structure 4

This chapter introduces the context, motivation, and the goals of this dissertation work. Finally, it presents the overall structure of this document.

1.1 Context

From its inception as a military experiment to becoming an integral part of everyday life, the Internet has come a long way. However, the journey is far from over. Now, we are entering the era of the "Internet of Things" (IoT), a time of even more pervasive connectivity in which anything and anyone, anywhere, are connected at anytime.

"IoT represents the next significant step in the Internet's evolution" [TM17]. It envisions a reality in which most devices will be connected to a network and interact with the physical environment through sensors and actuators, collecting and exchanging data.

Sensors gather data about the environment and once this data has been processed — either on the edge of the network or in a remote server —, some action is taken on the basis of the information extracted. That might involve directly modifying the physical world through actuators. These actions are often dependent on the state of the environment at a given point in time — in fact, this context awareness is one of the most important aspects of IoT [PZCG14].

IoT is a highly promising vision, and predictions show a rapid growth over the coming years both in terms of market value and the number of connected devices. According to the forecast in Figure 1.1, by 2020 the number of IoT connected devices worldwide will have grown to almost 31 billion worldwide, and reached up to 75 billion by 2025 [Sta01]. Although different forecasts

predict different growth rates, they all show a rapid growth over the coming years. Despite the fact that some predictions can be viewed as unrealistic, it is safe to estimate that the number of IoT connected devices worldwide will soon surpass the 10 billion mark [Nor16].

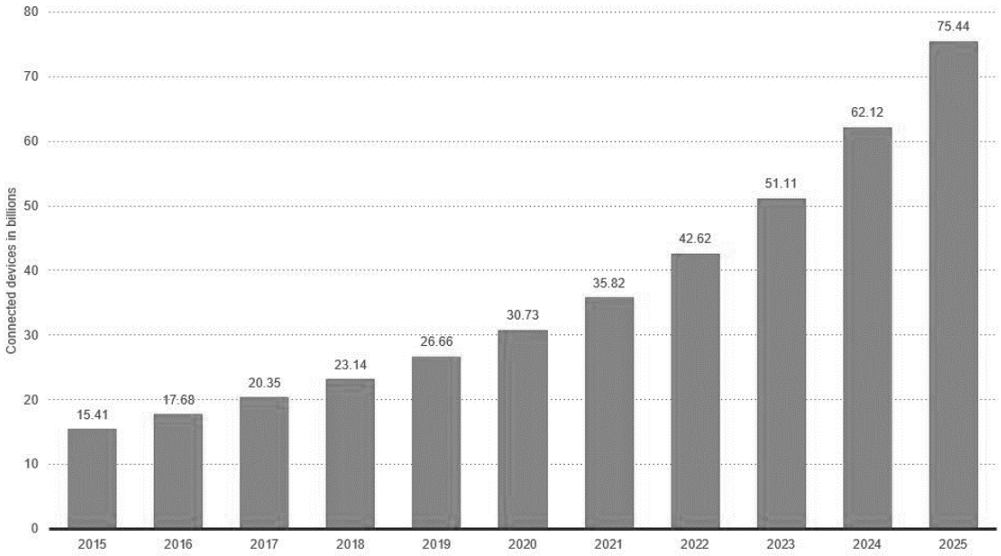


Figure 1.1: Evolution in the number of IoT connected devices worldwide from 2015 to 2025 [Sta01].

This technology is expected to impact our professional, personal and social environments, by allowing the development of new promising solutions in a variety of domains, including, but not limited to, agriculture, healthcare, utilities, and transportation, pushing towards the realization of other concepts, such as smart cities and smart homes [Tan10, KBG13]. However, its impact on healthcare will perhaps be its most important effect [Dim16].

In the not so distant past, health services turned to Electronic Health Records (EHRs), mobile health applications and electronic monitoring systems to improve the quality of patient care [VPZ12]. Nonetheless, these measures will likely prove insufficient to ensure the efficiency and safety of patient care in the future, as population aging and the increasing prevalence of chronic conditions put a strain on healthcare services.

The unprecedented aging of the population is a worldwide phenomenon — clearly noticeable in Figure 1.2 — that shows no evidence of slowing down. Additionally, a population growth of 4% is expected by 2050, a fact that will have profound implications in many aspects of everyday life [ESA01]. These demographic trends give rise to the need, on the one hand, to promote independent living and, on the other hand, reduce the increased pressure on health services.

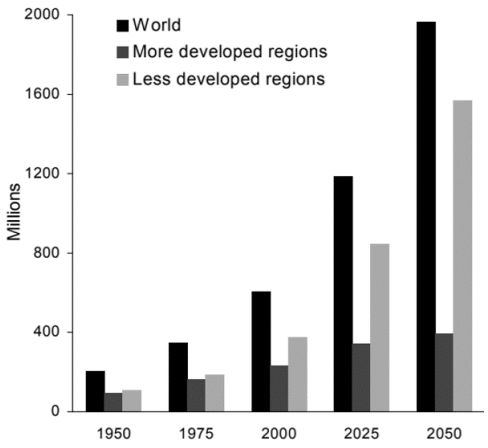


Figure 1.2: Population aged 60 or over: world and development regions, 1950-2050 [ESA01].

Moreover, the last half century observed a continuous increase in the incidence and prevalence of chronic conditions in most developed and developing countries [Wor01]. This wide range of chronic conditions includes HIV/AIDS, tuberculosis, cardiovascular disease, diabetes, and long-term mental disorders, and their treatment comes at a high cost to society, governments, and families, and puts an additional strain on healthcare services [Wor01].

If the medical community is able to overcome its reluctance to embrace new technologies [SM14], integrating IoT features into medical devices will greatly improve the quality and effectiveness of health services, making it possible to cope with the issues of generalized aging of the population and with the increase in the prevalence of chronic conditions, while bringing especially high value for those requiring constant supervision.

By 2020, health-related technology will account for 20% of IoT market, and is projected to reach an estimated value of 1.335 trillion dollars that same year [Col17]. Even today, body sensors are no longer gadgets used exclusively by athletes, runners, and joggers and are making their way into the general market, while several other wearable devices to monitor a person's health condition are being developed. With the increasingly common use of wearable devices such as smartwatches and fitness wristbands, we are already witnessing the convergence of healthcare and IoT, a trend which is expected to continue and, in addition to allowing a more independent life style, will revolutionize healthcare.

1.2 Objectives and Motivation

Current demographic and technological trends will likely drive to the emergence of IoT technology. Healthcare is only one of the domains that will benefit from the vast range of solutions IoT can provide. Thus, taking into consideration the range and scale of its applications, IoT will become an integral part of everyday life.

However, there are challenges associated with the development and testing of IoT applications and services, challenges that existing solutions have not yet properly addressed. Although there are already several tools that can be used in the testing of IoT systems, there are a number of faults that can be pointed out.

With the knowledge that failures in IoT applications can have dire consequences, the importance of ensuring their correctness becomes apparent.

Nonetheless, implementing test suites and test cases covering multiple aspects, interfaces, and protocols is a demanding and tedious task, and it is from this fact that stems the need for a comprehensive test framework for IoT automated integration testing which will allow to ensure a solution's conformity to the elicited requirements.

To address this issue, this work aimed to:

- Identify the short-comes of existing test solutions;
- Conceive a pattern-based approach that can be applied to various scenarios to test recurring behaviours in the scope of IoT, enhancing reuse;
- Develop a test framework for integration testing of IoT ecosystems that requires minimal technical knowledge and provides out-of-the-box functionality.

1.3 Document Structure

Beyond this introduction, this document is structured as follows:

- Chapter 2, *Background and State-of-the-Art*, provides some background on IoT and an overview of the state-of-the-art in the field of software engineering and testing for IoT, along with a technological review of some tools that fit this scope, focusing on the engineering challenges still to be tackled.
- Chapter 3, *A Feature Model and Test Patterns for IoT*, introduces a feature model for IoT ecosystems used in deriving a set of IoT test patterns, which are then further described.
- Chapter 4, *Izinto Test Framework*, describes the framework developed with respect to its functionality, its architecture, and implementation details.
- Chapter 5, *Validation*, describes the scenario considered for validating the framework — in terms of its features, components, and architecture — and details the various test cases covered.

- Chapter 6, *Conclusion*, concludes this dissertation with an overview of the work performed, outlines the contributions achieved throughout this dissertation, and lays out suggestions of future work.
- Appendix A, *Configuration File Example*, contains an example of a complete configuration file, corresponding to the settings for testing the scenario considered for the purpose of validating the framework proposed.
- Appendix B, *Scientific Publications*, encloses — in full — the scientific publications written within the scope of this work of dissertation.

Chapter 2

Background and State-of-the-Art

2.1 IoT Concepts, Architecture and Technologies 8
2.2 IoT for Healthcare Ecosystems 10
2.3 Software Development for IoT 13
2.3.1 Challenges 13
2.3.2 Existing Solutions 14
2.4 Testing for IoT 16
2.4.1 Challenges 17
2.4.2 Existing Solutions 18
2.5 Patterns and IoT 22
2.6 Summary 23

This chapter provides some background on IoT, with regard to key concepts, reference architectures, and relevant technologies. Further on, it covers the application of IoT in the field of healthcare, reflecting on future applications and on the challenges specific to this domain, and listing some existing application scenarios. Then, an overview of the state-of-the-art in the field of software engineering for IoT is presented, with a focus on the engineering challenges posed by the development and testing of IoT systems. Particularly, a technological review of some tools that fit this scope is carried out and some background is provided on design patterns for IoT and related work on pattern-based approaches to testing.

2.1 IoT Concepts, Architecture and Technologies

The expression "Internet of Things" was first introduced by Kevin Ashton in 1999, used as the title of a presentation on supply chain management [Ash09]. Since then, its definition has become more inclusive, spanning various application domains — including home automation, smart cities, social life and entertainment, health and fitness, smart environment and agriculture, supply chain and logistics, and energy conservation [SS17].

Several definitions have been proposed. It can be viewed as the interaction between the physical and digital worlds by means of sensors and actuators [VFG+09]. In a more convoluted way, it can be defined as the paradigm in which computing and networking capabilities are embedded in virtually any object and used to query and change the state of the object [ITU05].

The core infrastructure of an IoT framework is made up by sensors, actuators, servers, and the underlying communication network [SS17].

Although there are several architectures proposed in the literature (Figures 2.1 and 2.3) and there is no consensus on a single architecture for IoT, we can distinguish a general three-layered architecture [AFGM+15, IKK+15], as depicted in Figure 2.1, which captures the main idea of IoT, consisting of three distinct layers: a perception layer, a network layer, and an application layer.

The perception layer corresponds to the physical layer made up by the sensors used for sensing and gathering data about the environment. There are various types of sensors. They are usually of small size, have low cost, and low power consumption. The smartphone — which has embedded in itself several different sensors — can be considered the most popular sensing device [LML+10], but the use of other types of sensor is becoming increasingly common, such as it is the case of sensors for measuring temperature, pressure, humidity, physiological body parameters, and chemical and biochemical substances [SS17].

In turn, the network layer enables the connection between the various devices, allowing for the transmission and processing of sensor data. IoT network technologies include Ethernet, cellular and WiFi networks, as well as new networking technologies being developed specifically to meet the challenges of IoT, such as Low-Power Wireless Personal Area Networks (LPWANs), Bluetooth Low Energy (BLE), ZigBee, Near Field Communication (NFC), and Radio Frequency Identification (RFID) [Ger18].

Lastly, the application layer delivers application specific services. The application layer is therefore responsible for data processing and presentation. This layer can be based in different communication protocols for IoT devices, ranging from HTTP [FGM+99] to other novel protocols for IoT environments such as Message Queue Telemetry Transport (MQTT) [OAS14] and Constrained Application Protocol (CoAP) [SHB13]. For each scenario, the most appropriate protocol will depend on the specific requirements, as each protocol has its own strengths and drawbacks [Pat14].

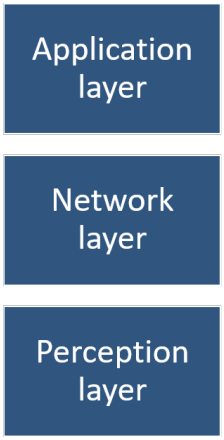


Figure 2.1: Three-layered architecture.

The ability to collect data from large numbers of sensing devices and the need for processing that data to obtain useful information as illustrated in Figure 2.2 makes Cloud Computing, Data Visualization, Data Analysis, Machine Learning, and other Big Data topics, research areas with great business potential [SLMN15].

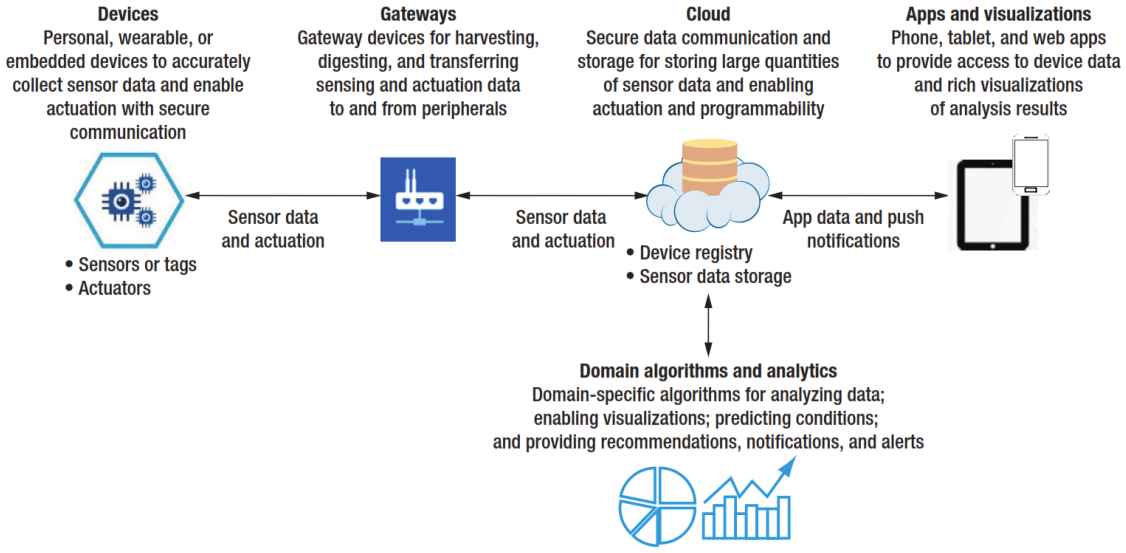


Figure 2.2: Detailed view of the emerging end-to-end IoT architecture, including its individual elements [TM17].

Although cloud computing services can scale in ways that meet IoT's storage and processing requirements, there are applications that require low latency [GVGB17]. That is the case of health monitoring, as delays introduced by the transfer of data between devices at the edge of the network and the cloud can significantly hinder performance [GVGB17]. Figure 2.3 shows the Edge-Fog Cloud architecture, which follows a Fog computing paradigm and where cloud services are extended to the edge of the network to decrease latency and network congestion [GVGB17], as proposed in [MK17]. The edge layer corresponds to an assortment of loosely coupled devices — the sensors, actuators, and other user devices. The fog layer resides on top of the edge layer and is made up by a set of networking devices — such as routers, and switches — able to run application logic. As a consequence, the core of the Edge-Fog Cloud architecture — the cloud — acts primarily as a repository for storing data.

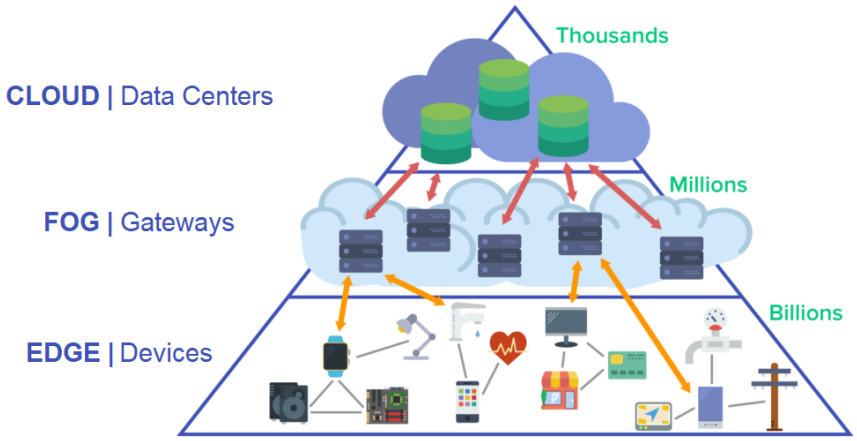


Figure 2.3: Edge-Fog Cloud architecture for IoT [Spo17].

2.2 IoT for Healthcare Ecosystems

IoT applications are expected to impact several domains, but its impact on healthcare will perhaps be its most important effect [Dim16]. Healthcare is a far-reaching field, encompassing personal healthcare, the pharmaceutical industry, healthcare insurance, robotic prosthetics, and bio-sensors, and, therefore, the list of potential IoT applications in this area is endless [iS].

In [IKK+15], a comprehensive survey of IoT for healthcare is presented, reviewing the state-of-the-art network architectures, applications, and trends in IoT healthcare solutions.

In [MHAK17], a scenario consisting of a Wireless Body Sensor Network (WBSN) and a smart health gateway is used to analyse the issue of IoT security from the healthcare perspective, and confidentiality, integrity, and availability are listed as key security requirements.

The challenge imposed by the heterogeneous nature of IoT systems is the subject of [YD11], in which is demonstrated that medical device data compliant with ISO/IEEE 11073 — a group of standards addressing the interoperability of personal health devices [Sta10] —, can easily be transformed into schemes compliant with HL7 — a set of standards for the exchange and storage

of electronic health information [Hl08]—, facilitating the interoperability of clinical applications and medical devices.

For now, the two main IoT use cases for healthcare are the remote health monitoring and the tracking, monitoring, and maintenance of assets using RFID technology [iS]. Only by combining IoT solutions with applications and technologies in the field of robotics [GRC+14], artificial intelligence [ALNT14], and data mining [TLCY14] will it be possible to leverage the full potential of IoT.

Health sensing components are becoming ever more compact and portable, allowing them to be worn round the clock, continuously monitoring and recording health conditions and to trigger alarms in case any abnormality is found so that proper actions can be taken. The data collected can also be used to create detailed EHRs, with all the medical details of a person, generating statistics which enable the surveillance and risk mapping of some diseases [SK16].

Remote Patient Monitoring (RPM) and Ambient-Assisted Living (AAL) are currently the main focus of IoT research in the field of healthcare. RPM and AAL systems make independent living possible for the elderly and enables patients with chronic or serious health conditions to avoid a prolonged hospital stay or reduce the number of visits to medical facilities, by allowing them to be monitored and advised from anywhere.

Because most IoT research in the field of healthcare has been focused on improving the quality of care through remote health monitoring solutions [iS], there is already some work in the area of RPM and AAL, presenting solutions designed for specific deployment scenarios (Table 2.1).

Project	**Deployment Scenario**	**Communications**	**Standards**
Yang et al. [YXM+14]	Home	Wi-Fi, ZigBee	N/A
Rahmani et al. [RTG+15]	Home, Hospital	Bluetooth, Wi-Fi, 802.15.4/6LoWPAN	IEEE 11073
Saponara et al. [SDFC16]	Home, Pharmacy Nursing Home	BLE, Wi-Fi, 3G	HL7 CDA
Miranda et al. [MCW+16]	Home, Nursing Home	NFC, Ethernet, Bluetooth	Continua Health Alliance [1]

Table 2.1: Related projects in the area of remote health monitoring: summary of deployment scenarios and technologies.

[1] Set of guidelines put forward by PCHAlliance, based on accepted standards including IEEE 11073 and HL7 [Org].

Yang et al. [YXM+14] presented an intelligent home-based platform — the iHome Health-IoT. It involves a medicine box — iMedBox —, a pharmaceutical packaging — iMedPack —, and a wearable bio-medical sensor device — Bio-Patch. The proposed platform allows the seamless integration of in-home healthcare devices and services, and is enabled by Wi-Fi, ZigBee, and RFID.

Rahmani et al. [RTG+15] proposed a conceptual gateway aimed at healthcare applications for hospitals and private homes — the Smart e-Health Gateway —, with support for several communication technologies such as BLE, Wi-Fi, and 6LoWPAN. Their work also included the development of a proof-of-concept IoT remote health monitoring system for gathering and storing electrocardiography (ECG) data collected using wireless sensors. It focused on the implementation of the actual gateway and on technical aspects related to energy efficiency, scalability, interoperability, and reliability issues.

Saponara et al. [SDFC16] introduce a remote healthcare model, which exploits wireless bio-medical sensors and a local gateway, responsible for sensor data acquisition, sending the data to a remote e-Health service centre after processing it to provide statistics and alerts about possible medical conditions. It is directed at the continuous and long-term monitoring of patients affected by chronic illness, and can be deployed either at the patients' home, or in a healthcare unit.

Miranda et al. [MCW+16] present a platform — the CRIP — which aims at supporting care-givers and citizens to manage health routines. NFC and fingerprint bio-metrics are used for identification and authentication, while Bluetooth and Ethernet enable communication with health devices and web services for wider integration with other platforms.

2.3 Software Development for IoT

2.3.1 Challenges

In the age of IoT, everyday devices are becoming connected, remotely and dynamically programmable. It is important to understand that IoT development differs from most typical mobile and web applications in several aspects, something that IoT developers must take into account.

The paradigm shift brought on by the IoT is making apparent that today's development methods, languages, and tools are poorly suited to face the challenges and technical issues that arise from the emergence of millions of programmable things in our everyday life. The work of Taivalsaari and Mikkonen [TM17] highlights such challenges — which are summarily presented next —, providing a road-map to this new "programmable world".

The first of these challenges pertains to the distributed, and always-on nature of IoT systems. From the software developer's viewpoint, the possibility of device failure, and intermittent and often unreliable network connections require constantly preparing for failure. Though these characteristics are not new to software developers, they require them to program in a fault-tolerant and defensive manner. In doing so, developers incur the danger of writing too much code to handle potential errors and exceptions, thus making these systems much more difficult to maintain. Ideally, programming languages should make it easier to balance application logic and error handling.

The inadequacy of programming languages and development tools is a problem, as IoT development currently relies, for the most part, on mainstream programming languages, such as C, C#, Java, JavaScript, or Python. In fact, due to their popularity in web development, Node.js and JavaScript are becoming prominent tools for IoT development, even though not designed for writing asynchronous, distributed applications or for programming-in-the-large.

A related issue is that of multi-device programming. IoT devices are part of large, and highly dynamic systems, which must be managed and maintained. However, facilities for orchestrating such systems and mechanisms to allow flexible code migration are not widespread, making it harder to make development-and-deployment a standard practice for automating software delivery.

With IoT systems being an amalgamation of devices with varying computing power, storage capabilities, network bandwidth, and energy requirements, heterogeneity poses yet another major challenge. If, on the one hand, interoperability of such diverse devices calls for well-defined standards, on the other hand, it may be desirable to explore this diversity so as to develop solutions which optimize efficiency, for instance, in terms of power consumption or network usage.

Last but not least, security is a concern of paramount importance. By advancing the way we monitor and track ourselves and the things around us, what we do with the data collected — and how it is sent across networks — can get sensitive. Because IoT creates such unique challenges to privacy, end-to-end security should be considered, including authentication and encryption. However, concerns will no longer be limited to privacy and the protection of sensitive information, since remote actuation and programming capabilities can pose an even higher threat to safety.

2.3.2 Existing Solutions

At the moment, no universal software development environments exist that will allow developers to effortlessly write one IoT application that runs on all types of devices, nor for the orchestration and management of such large, and complex systems [TM17].

From an architectural stand point, a variety of design patterns have been observed and documented, establishing patterns that represent some key aspects of the IoT domain [CHA16]. Still with respect to design, the ThingML — Internet of Things Modelling Language — is inspired by the idea of the Unified Modelling Language (UML), and aims to address the challenges of distribution and heterogeneity in IoT, whilst facilitating collaboration between service developers and platform experts [MHF17]. Specifically, it allows writing code in a platform-independent way and compiling it to different platforms. In addition, it is possible to write platform specific components and link them to exiting Application Programming Interfaces (APIs) or libraries. This approach has evolved over the past years and it has been applied to different domains, including the case of a commercial e-health application for fall detection [MHF17].

For development purposes, there are a number of tools to support the development of IoT solutions, designed to help developers explore IoT devices and applications [Smi]. These include IoT operating systems, IoT hardware platforms, IoT middleware tools, and IoT platforms [Smi].

Lightweight operating systems — such as Contiki [DGV04], and more recently RIOT [BHG+13] —, have been developed and used as software platforms upon which it is possible to implement network stacks and applications running on IoT devices [RWBO15].

IoT hardware platforms include development boards such as Arduino, Espruino, Intel's Galileo, and Tessel [TM17].

A key technology in the realization of IoT systems is middleware. Middleware platforms abstract hardware details by providing an API for communication, data management, computation, and security [SS17]. A number of middleware solutions are currently available, such as Oracle's Fusion Middleware [2], OpenIoT [3], and Kaa [4].

OpenIoT is an open-source platform which enables the integration of IoT data and applications within cloud computing infrastructures, by allowing the deployment of and secure access to semantically inter-operable applications [SKH+15].

Kaa is a multi-purpose middleware platform for IoT that allows building complete end-to-end IoT solutions in an expedited manner [Tecb]. It provides an open-source, feature-rich toolkit for IoT development, offering a set of out-of-the-box IoT features that can be easily used to implement various IoT use cases [Tecb].

[2] https://www.oracle.com/middleware/
[3] http://www.openiot.eu/
[4] https://www.kaaproject.org/

IoT platforms can be divided into two categories: the platforms which provide cloud services for IoT solutions — such as IBM Watson [5], Azure [6], and AWS [7] —, and the tools which support the development process — as it is the case of Eclipse IoT [8], Node-RED [9], and Flogo [10].

Eclipse IoT provides technology to build IoT devices, gateways, and cloud platforms [Smi], encompassing a number of projects and tools with focus on the development of IoT systems [Foub].

Node-RED is a prime example of a visual approach to software development. It is an Integrated Development Environment (IDE) which allows to quickly connect a wide range of hardware devices and software to web services [BL14]. It is a flow-based programming tool that makes it easier to create even very advanced applications by wiring together graphical blocks — called nodes. Initially developed as an open-source tool at IBM in 2013, it has gained popularity as a general purpose programming tool for IoT [CCH15], and there is a growing number of open-source software components corresponding to different nodes.

Flogo is an ultra-lightweight development framework based on the Go programming language, capable of being run on a multitude of platforms — edge devices, gateways, cloud services — and highly optimized for unreliable IoT environments [Wäh16]. Flogo supports the development of IoT applications by integration and allows developers to work either by writing source code or by using the interface for visual coding, debugging, and testing [Wäh16].

In addition to the previous development tools already described, **MySignals** [11] was also identified, this one being specifically dedicated to the development of IoT healthcare applications. MySignals is a development platform for medical devices and healthcare applications. It allows the addition of several sensors to build custom solutions, measuring various different biometric parameters. This is a proprietary tool, and its use implies the subscription of a paid plan.

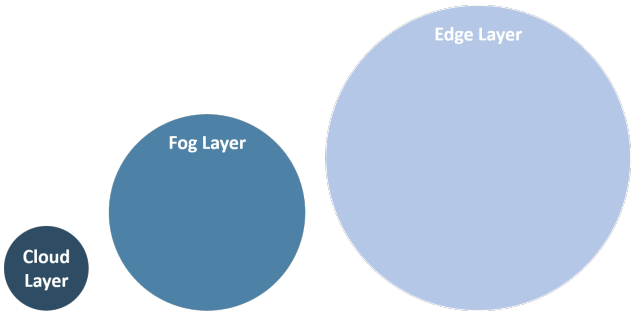
[5] https://www.ibm.com/watson
[6] https://azure.microsoft.com/
[7] https://aws.amazon.com
[8] https://iot.eclipse.org
[9] https://nodered.org
[10] http://www.flogo.io/
[11] http://www.my-signals.com

2.4 Testing for IoT

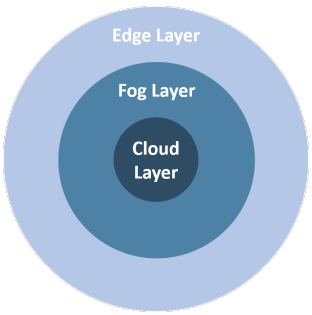
Testing is an important part of the software development process, being crucial to ensure a system's compliance to both functional and non-functional requirements. In fact, verification and validation activities are complex tasks and can amount to up to 75% of the development costs [HS02], making testing automation essential.

The practice of merging code and automatically running a build process including tests in production-like environments defines the concept of Continuous Integration (CI) [Guc], which has emerged as a best practice for software development, being used extensively in the area of web development, and increasingly so in the development of mobile applications [Mer06]. For CI to be applied to IoT development, solutions for test automation will be required.

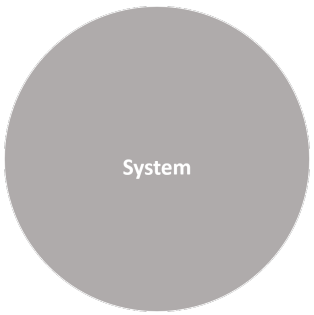
There are still no established good practices when considering a test strategy specific for IoT [Ger17]. Typically, the first level of testing is the testing of individual units — Unit Testing (UT) — , which can be applied to the test of the functioning of the components of each of the several layers of IoT (Figure 2.4a). The next level of software testing is Integration Testing (IT), where individual units are combined and tested as a group with the purpose of exposing faults in the interaction between the units being tested [Fun], and it is critical to assure the proper interoperation between the different layers of IoT (Figure 2.4b). Finally, System Testing (ST) should be performed in a manner transversal to all layers (Figure 2.4c), so as to evaluate the system's compliance with its specified requirements.



(a) Breakdown of IoT applications into different layers for Unit Testing.



(b) Combination of different IoT application layers for Integration Testing.



(c) Abstraction of IoT applications for System Testing.

Figure 2.4: Levels of testing across IoT layers.

In the context of IoT development, IT is of particular importance, as the definition of testing scenarios that go beyond checking separately the functionality of the individual components of the System Under Test (SUT) is crucial to ensure that not only does each component behave as expected, but they also work together as an IoT solution [Ger17].

2.4.1 Challenges

A number of challenges need to be overcome before the requirement of test automation is met and it becomes possible to achieve CI in the IoT [Mer06]. The effort and potential problems associated with the tasks of debugging and testing is likely to be underestimated by the vast majority of developers, who are not used to dealing with such issues, even though some are not necessarily new [TM17]. Understanding the specifics of IoT and analyzing the most important technical challenges are two crucial steps when preparing a testing strategy [Ger17].

As pointed out in [Pra17], testing IoT systems means performing end-to-end testing, and may imply carrying out:

- **Functional Testing** to evaluate the SUT's conformance to requirements;
- **Compatibility Testing** i.e., verifying and validating the possible combinations of communication protocols and devices;
- **Usability Testing** so as to verify the SUT's ease of using experience;
- **Network Testing** in order to validate the SUT with different network conditions;
- **Security Testing** which includes verifying the SUT's handling of data privacy;
- **Performance Testing** to ensure scalability and the overall performance of the SUT, for instance with respect to power consumption, and memory usage.

Non-functional requirements, namely those related to interoperability, connectivity, scalability, and security, tend to be difficult to test in a systematic way [Mer06]. IoT systems rely on various closely-coupled hardware and software components. In order to ensure interoperability, it is necessary to test all possible software and hardware combinations, as it is not enough to adhere to protocol specifications [RWBO15]. Another issue is that, because network connection plays a vital role in the concept of IoT, these systems need to be tested in various network conditions — a testing methodology which takes into consideration various connectivity scenarios while focusing on ways to deal with security threats is required [Bis17].

Given the extremely dynamic nature of IoT systems, the debugging and testing these systems — which often consist of a large number of devices, usually very distinct, and inserted into complex real-world environments — can be very challenging. This task can be made even more complex when the SUT has the ability to self-adjust and balance computation resources, network bandwidth, and battery consumption by re-allocating computation dynamically between its components [TM17].

Another difficulty has to do with the need for real time data and the interaction between the SUT and the environment. On the one hand, there is the problem of test data: exhaustive test data can be difficult to obtain for IoT projects, since it depends on real-world situations [Mer06]. Replicating these settings is not always possible, and so generating test data can be an alternative to collecting it from a real-world scenario. On the other hand, there is also the problem of simulating the actuation itself, since having physical actuators in a test environment is often not feasible. These issues become even more problematic when the SUT is related to healthcare [Bis17].

While some tests can be conducted on device emulators or simulators, other tests might need to be performed on real devices, in order to ensure that the combination of software and hardware works as specified [RWBO15]. Hence, it may be necessary to perform these tests in a network consisting of real IoT devices, in a setup similar to that of the SUT.

In conclusion, testing automation of both software and hardware components is required to enact an automatic approach to the testing process and achieve systematic, and regular release processes.

2.4.2 Existing Solutions

The question of "what tools and frameworks can be used to efficiently test IoT systems" [RWBO15] is relevant because, as seen, testing IoT applications is a complex task. Even so, up until now, both research and industry have overlooked the topic of IoT testing to some extent [RWBO15].

Three distinct methodologies can be followed in IoT verification and validation: formal verification — proving or disproving the correctness of the SUT with respect to its specification, using formal methods —, simulation — tools that allow designing, creating and testing IoT systems without actually using real IoT devices —, and pilot testing — which consists in deploying the SUT in a test environment as close as possible to the environment in which the system will be deployed.

The outline of a framework for automated testing must include protocol simulators, data collection capabilities, and virtualization [Pra17]. On the one hand, in the IoT the SUT is inherently networked [RWBO15], and therefore it is important to test the communication between the various device endpoints and interfaces under different network settings, and evaluate how the SUT will perform with respect to test metrics such as loss, delay, error, and quality of service. On the other hand, virtualization typically includes simulation tools, which enable the design, creation, and testing of IoT applications without actually using real IoT devices, minimizing the dependency on a real-time environment for testing.

Some software development and testing tools — such as ArduinoUnit [Mur13] and PlatformIO [Pla] — can be used for testing IoT applications, often relying on physical devices to conduct the testing, but focus mainly on the level of Unit Testing.

PlatformIO is an open-source IDE for IoT which offers several tools, including a cross-platform compiler, a library manager, a debugger, and a UT system. It makes it possible to perform UT only within the edge layer, specifically at the level of the individual physical devices.

ArduinoUnit is a UT framework for Arduino projects, which limits its application to performing UT at the level of the individual physical devices within the edge layer [Mur].

Following an approach that resorts to a combination of model-based testing (MBT) techniques and service oriented solutions, Ahmad et. al conceived **MBTAAS** [ABF+16] which allows to systematically test IoT and data platforms.

Another possible approach to test IoT-based applications consists in using systems that behave like the SUT — emulators and/or simulators. **TOSSIM** [LL03] is a wireless sensor network simulator, built with the specific goal of simulating TinyOS devices, supporting two programming interfaces — Python and C++ —, and allowing various levels of simulation, from hardware interrupts to high-level system events, such as packet arrivals.

Similarly, **COOJA** [ÖDE+06, ANR] is an extensible Java-based simulation/emulation platform developed for the Contiki operating system. It provides a complete simulator for sensor node software, where the network, operating system and even the machine code instruction set are simulated, allowing developers to test their code and systems before running it on the target hardware, allowing UT and IT of edge devices — both physical and simulated. COOJA has support for radio link simulation, with recently added support for a more complex radio interference simulation from Wi-Fi and Bluetooth.

Looga et al. presented an emulation platform for IoT — **MAMMotH** [LODYJ13]. Its architecture presumes three distinct scenarios based on General Packet Radio Service (GPRS) communications. This is part of an ongoing project, with the ultimate goal of supporting nodes in the scale of tens of millions, and there are questions that are still open and need to be investigated in the future.

iFogSim is a simulator to model IoT and Fog environments which supports IT over virtualized devices — from within both the Edge and Fog layers —, but it focuses primarily on measuring the impact of resource management techniques in terms of latency, network congestion, energy consumption, and cost [GVGB17, Gup].

IoTIFY is a comprehensive IoT simulation solution that makes it possible to develop and test IoT solutions in the cloud, by simulating large scale IoT installations in a virtual IoT lab [S.L]. It allows performing UT as well as IT using virtualized hardware and by generating traffic in real time via HTTP, MQTT, or CoAP to test a platform for scale, security, and reliability issues. This is however a commercial tool.

SimpleIoTSimulator is a simulator for creating IoT environments made up of a large number of sensors and a gateway, supporting several of the common IoT protocols, including HTTP, MQTT, and CoAP [Sim]. Though it is aimed at IT, the SimpleIoTSimulator does not allow the modelling of Fog environments where services can be deployed both on edge and cloud devices [GVGB17].

To make it possible to test IoT systems as a whole, work has been pursued towards the development of IoT testbeds. A number of physical testbeds are already active and publicly available [GKN+11]. Although these tend to cover multiple IoT layers, they usually focus on a specific domain of application, and creating and maintaining these systems can be both complex and expensive.

FIT IoT-LAB is a heterogeneous testing environment, consisting in an open testbed made up by various sensors available for experimenting with large-scale wireless IoT technologies [IL, ABF+15]. It allows performing UT and IT remotely in a set of physical devices but it covers a limited number of IoT use cases and applications.

Table 2.2 presents a summary comparison of the IoT tools described above. Despite the fact that there are several solutions for testing IoT systems, following different testing approaches and focusing different test levels, it is possible to identify several limitations in existing solutions. For a number of solutions presented in the literature — as it is the case with MBTAAS —, it is not possible to find actual tools or frameworks to support implementation, making it complex to adopt and use them. Other tools — including TOSSIM, and COOJA — focus on a specific platform, language, or standard, failing to account for the heterogeneity of the IoT field. Furthermore, most do not allow for the possibility of improvement or functionality extension — for instance, by not making source code publicly available —, and do not provide out-of-the-box functionality.

Tool	IoT Layer	Test Level	Test Environ.	Prog. Languages	Supported Platforms	Scope	License
PlatformIO	Edge	UT	Physical Device	C,C++, Arduino	Multiple	Commercial	Closed
ArduinoUnit	Edge	UT	Physical Device	Arduino	Arduino	Academic, Commercial	Open
MBTAAS	Edge, Fog, Cloud	UT, IT, ST	Platform	OCL	N/A	Academic	N/A
TOSSIM	Edge	IT	Simulation	Python, C++	TinyOS	Academic	Open
COOJA	Edge	IT	Emulation	C	ContikiOS	Academic, Commercial	N/A
MAMMotH	Edge, Fog, Cloud	IT, ST	Emulation	N/A	N/A	Academic	N/A
iFogSim	Edge, Fog	IT, ST	Simulation	Java	N/A	Academic	Open
IoTIFY	Edge, Fog, Cloud	UT, IT, ST	Simulation	N/A	N/A	Commercial	Closed
SimpleIoT Simulator	Edge, Fog	IT	Simulation	N/A	N/A	Commercial	Closed
FIT IoT-LAB	Edge, Fog, Cloud	UT, IT, ST	Physical Testbed	N/A	Multiple	Academic, Commercial	Open

Table 2.2: Summary comparison of available IoT testing tools.

2.5 Patterns and IoT

Patterns describe a problem and then offer a solution [CHA16].

The technological aspects of IoT systems and the details of their implementation reveal recurring solutions for the common problems of the field, which can be described as design patterns. Although there is still a large number of recurrent solutions in the scope of IoT yet to be documented, efforts have already been made towards the definition and categorization of design patterns for IoT systems [CHA16]. Initial contributions in the context of defining these design patterns established patterns representing some key aspects of the IoT domain were made by Reinfurt et al. [RBF+16, RBF+17]. Further work has been pursued in defining patterns that address how to deal with highly-changeable, error-prone and failing devices, and their networks [RDD+17].

If, on the one hand, design patterns help architects and designers build IoT applications of varying levels of complexity [CHA16] and address some of the challenges to the development of IoT solutions, on the other hand, pattern-based approaches have been proposed for testing applications in other domains [PSCW10]. These approaches are based on the assumption that systems similar in design, i.e., that follow the same set of design patterns, should share a similar test strategy.

Siddiqui and Khan [SK17] proposed a pattern-based technique for testing cloud applications and identified a set of test patterns by studying what features this type of application would support. For the purpose of evaluation, they considered threats to cloud applications and discussed the applicability of these test patterns.

Moreira et. al [MPM13] put forward a pattern-based approach for Graphical User Interface (GUI) modeling and testing in order to systematize and automate the testing process. The notion of User Interface (UI) test pattern is introduced, corresponding to the association of a set of test strategies to a UI pattern. Each UI test pattern may be instantiated so as to describe the slightly different implementations and check if a test is passed. Recognizing the need to develop test strategies specific for the mobile world, Costa et. al [CPN14] performed an experiment to assess if the same approach could be used to test mobile applications, which produced encouraging results. Morgado and Paiva [MP15] extended the set of UI test patterns previously mentioned by identifying additional UI test patterns specific to the testing of mobile applications.

2.6 Summary

More than a concept, IoT is an architectural framework for integration and data exchange between the physical and digital worlds over an underlying communication infrastructure.

IoT applications span several domains. In the case of healthcare, the focus is on the improvement of care, with some work already existing in the area of RPM and AAL, presenting solutions designed for specific deployment scenarios.

Nonetheless, there are several challenges to realizing IoT, such as scalability and interoperability, which are related to the plurality of devices, networks, and sensors involved, the various communication protocols employed and integration requirements.

If, from a technical point of view, these challenges create the need for new hardware development, new communication protocols and technologies, and new techniques to process and analyze the huge amounts of data collected, from a software engineering perspective there is also the demand for new development technologies, processes, methodologies, and tools.

The difficulty to test IoT systems as a whole — due to the number and diversity of individual components and the distributed nature of the system —, along with the difficulty to test said components individually — due to the dependency on other components — [LF16], gives rise to the need for an efficient and effective way to implement automated testing in IoT.

Test automation in IoT applications will make it possible to test various devices and services spread across different IoT layers, and in doing so, arrive at a common framework for fast software development. Although there has been some work put into the definition of patterns for IoT, no references to pattern-based approaches to testing IoT were found in the literature, even though this methodology has been explored in other domains.

Although there are already several tools that can be used in the testing of IoT systems, a number of issues can be pointed out, such as focusing on a specific platform, language, or standard, hindering the possibility of improvement or extension, and not providing out-of-the-box functionality.

Chapter 3

A Feature Model and Test Patterns for IoT

3.1 Feature Model **26**
3.2 Test Patterns **28**
3.2.1 Test Periodic Readings 29
3.2.2 Test Triggered Readings 31
3.2.3 Test Alerts 33
3.2.4 Test Actions 35
3.2.5 Test Actuators 37
3.3 Summary **38**

This chapter explains the approach used to arrive at a set of Test Patterns for IoT ecosystems. It introduces a feature model for IoT ecosystems used to identify a set of IoT test patterns, which are then further described.

3.1 Feature Model

By its own nature, the field of IoT involves a wide variety of systems and devices with very diverse features and, consequently, any testing approach must take into consideration these differences.

The components of an IoT ecosystem can, generally speaking, be categorized and grouped according to their characteristics. Because the test of IoT systems is often closely related to these features, it is useful to devise a model that allows representing the plurality of components and their features within an IoT ecosystem. By identifying the different components of an IoT system and classifying each component in terms of its features, it is possible to determine which functionality must be tested and to select which test(s) pattern(s) should be applied accordingly. The model depicted in Figure 3.1 is loosely based on the formalism of feature models [Bat05] and considers four categories and subcategories:

- **Local Device**

  A local device can be either a sensor or an actuator. A sensor can be categorized in accordance to its reading mode, i.e., whether readings are performed automatically, in a periodic fashion and without human interaction, or, conversely, if readings are triggered by a user.

  Additionally, it is possible to differentiate sensors based on the way its data is transmitted — actively or passively. A sensor can actively send readings or keep data which must be retrieved. The ability of a sensor to measure various parameters as opposed to a single parameter is yet another differentiating factor.

- **Hosting Device**

  The category of hosting devices corresponds to gateways, which can be defined by features such as reading, processing, and publishing data, and potentially trigger actions (either each time a condition is met, or each time a condition status changes).

- **Remote Server**

  A remote server commonly handle communications and data persistence and may also support the triggering of actions.

- **External Application**

  External applications can support data visualization (either real time data or an historic record), and the generation of alerts (either each time a condition is met, or each time a condition status changes).

It is worth noting that this is only a partially-defined model, as there may be other characteristics and/or features that were not incorporated.

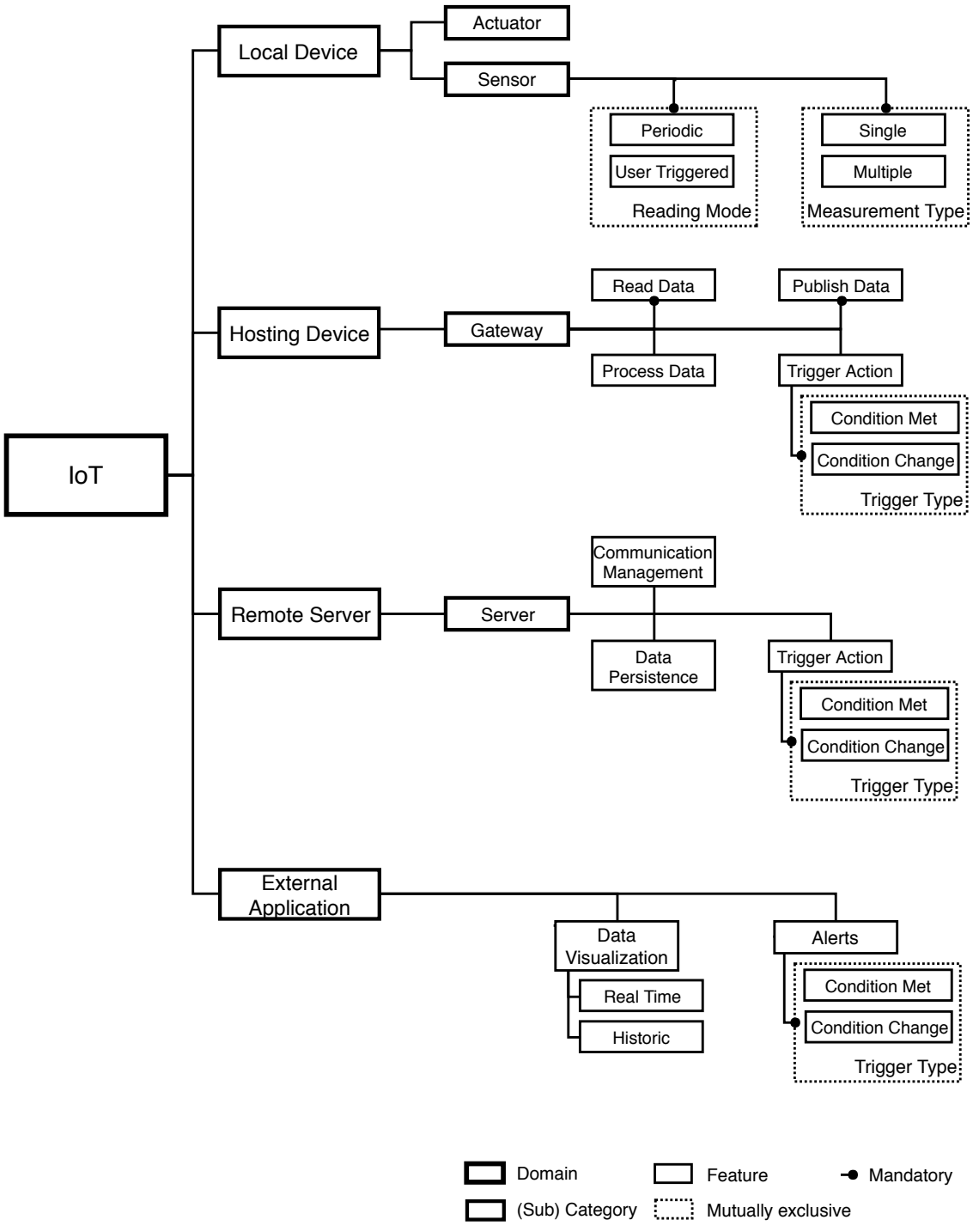


Figure 3.1: Feature model for an IoT ecosystem.

3.2 Test Patterns

Taking into consideration the particularity of IoT systems, it is possible to recognize a set of test strategies to test recurring behaviours of IoT systems which can be described as test patterns. Because the implementation of a given functionality may differ across different IoT ecosystems, a given test pattern may be configured so as to describe slightly different implementations and verify if the various checks defined passed or failed.

Within the scope of this dissertation, five different test patterns were identified, as listed in Table 3.1. These were derived from the feature model described in Section 3.1, and reflect both key features of several system components and overall functionality that an IoT ecosystem commonly supports.

Pattern	Description
Test Periodic Readings	Check whether a sensor is capable of performing readings at a fixed rate and that those readings are correctly transmitted and persisted within the IoT system
Test Triggered Readings	Check whether a sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system
Test Alerts	Check whether alerts are sent to pre-set subscribers whenever a pre-set condition is met
Test Actions	Check whether pre-set actions are executed whenever a pre-set condition is met
Test Actuators	Check whether an actuator is capable of executing commands, changing its state accordingly

Table 3.1: Test patterns identified.

In order to make these patterns reusable and to facilitate the definition of more patterns, a template based on the one proposed by Meszaros and Doble [MD97] was used. Each pattern is distinguished by a name and further described through various fields, defined as follows:

- ***Name***: unique identifier to shortly refer the pattern;
- ***Context***: situation where the problem occurs;
- ***Problem***: description of the problem addressed by the pattern;
- ***Solution***: description of the proposed solution for the pattern;
- ***Example***: example where the (IoT Test) pattern can be applied.

3.2.1 Test Periodic Readings

Name:

Test Periodic Readings

Context:

Under the paradigm of IoT, all *things* will be connected to the Internet and interact with the physical environment through sensors, collecting and exchanging data. Being equipped with sensing capability, these devices can periodically perform measurements on a number of parameters and transmit this data — either actively or passively, depending on their degree of autonomy.

Problem:

It must be ensured that a sensor is capable of performing readings at a fixed rate and that those readings are correctly transmitted and persisted within the IoT system, under normal operation.

Therefore, it is necessary to:

- Check that the readings are performed with the pre-set periodicity, with acceptable deviations;
- Check that the readings are transmitted with a delay below a pre-set maximum;
- Check that the readings are correctly transmitted;
- Check that the readings are valid and, optionally, that variations are observed.

Solution:

The test pattern for this type of behavior is only applied when a sensor supports the periodic execution of readings and consists of the following steps:

1. Setup the sensor(s);

   This may include specifying the type of readings to be performed and setting an adequate rate for each type.

2. Setup a gateway (optional);

   This may include configuring it to maintain a log of the data collected, so that it can be compared with the data persisted.

3. Induce some variation in the parameter(s) being measured (optional);

   In some cases, it may be possible to use an actuator to produce those variations. In the cases in which that is not feasible, it may be possible to resort to human input to produce those variations.

4. Verify if valid readings are persisted within the IoT system, at the expected rate, accounting for possible transmission delays.

   The validity of the readings relates to their value — specifically, whether pre-set thresholds and/or trends are observed — and to the consistency across the several points of the IoT system considered. Checking if the readings are performed with the pre-set periodicity and accounting for possible deviations and transmission delays may be done by checking the timestamps of reading and reception, assuming that the different components' clocks are synchronized.

It should be noted that the total runtime for the test must be defined taking into account the periodicity of the readings, so as to ensure that a sufficient number of measurements are performed.

Example:

Take an IoT system which consists of temperature and air quality sensors, which are setup to periodically perform measurements of the environment the system is in. The temperature sensors are expected to perform temperature readings every five minutes and are capable of measuring temperatures from -40^{o}C up to 80^{o}C. The air quality sensors measure the level of Carbon Monoxide (CO) and Carbon Dioxide (CO_2) every minute, in Parts Per Million (PPM). Using this pattern it becomes possible to ensure that all sensors perform these readings at the expected rate and that those readings are correctly transmitted and persisted within the IoT system. This pattern can be instantiated simply by providing a set of inputs, including the test runtime, sensor settings — with regard to id, type of readings performed, expected periodicity, acceptable deviation and delay, and expected value ranges —, and the means for accessing the data collected.

3.2.2 Test Triggered Readings

Name:

Test Triggered Readings

Context:

IoT devices are meant to work in concert for and with people at home, in industry or in the enterprise. In a number of situations, users are expected to trigger readings, commanding the sensor to perform measurements on a number of parameters and to transmit this data — either actively or passively, depending on their degree of autonomy.

Problem:

It must be ensured that a sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system, under normal operation.

Therefore, it is necessary to:

- Check that the readings are transmitted with a delay below a pre-set maximum;
- Check that the readings are correctly transmitted;
- Check that the readings are valid and, optionally, that variations are observed.

Solution:

The test pattern for this type of behavior is only applied when a sensor supports the triggering of readings and consists of the following steps:

1. Setup the sensor(s);

   This may include specifying the type of readings to be performed.

2. Setup a gateway (optional);

   This may include configuring it to maintain a log of the data collected, so that it can be compared with the data persisted.

3. Trigger the readings;

   In some cases, it might be possible to use an actuator to trigger the reading, or — in the cases in which that is not feasible —, have a human do it.

4. Induce some variation in the parameter(s) being measured (optional);

   In some cases, it may be possible to use an actuator to produce those variations. In the cases in which that is not feasible, it may be possible to resort to human input to produce those variations.

5. Verify if valid readings are persisted within the IoT system, at the expected rate, accounting for possible transmission delays.

   The validity of the readings relates to their value — specifically, whether pre-set thresholds and/or trends are observed — and to the consistency across the several points of the IoT system considered. Checking if the readings are performed and transmitted within the maximum acceptable delay may be done by checking the timestamps of reading and reception, assuming that the different components' clocks are synchronized.

It should be noted that the total runtime for the test must be defined taking into account the time necessary to trigger the readings, and in a way that allows a sufficient number of measurements to be performed.

Example:

Take an IoT system which includes a blood pressure monitor, which can be used to check diastolic and systolic pressure, as well as heart rate. Using this pattern it becomes possible to ensure that this sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system, by providing a set of inputs, including the test runtime, sensor settings — with regard to id, type of readings performed, acceptable delay, and expected value ranges —, and the means for accessing the data collected.

3.2.3 Test Alerts

Name:

Test Alerts

Context:

Many IoT applications involve generating alerts whenever a certain condition is met. These alerts and conditions are usually dependent on the data collected by the sensors.

Problem:

It must be ensured that an alert is triggered and that all subscribers are notified, under normal operation.

Therefore, it is necessary to:

- Check the conditions for triggering the alert are met;
- Check that a notification reaches all the subscribers, within an acceptable delay.

Solution:

The test pattern for this type of behavior is only applied when an alert is expected to be generated when a specific condition — or set of conditions — dependent on readings collected by one or more sensors is met. It consists of the following steps:

1. Generate readings capable of triggering the expected alert(s);

   This can be attained either by using physical sensors and possibly actuators to produce readings within certain thresholds. This would involve:

   (a) Setup the sensor(s);

   This may include specifying the type of readings to be performed.

   (b) Setup a gateway (optional);

   Alternatively, this could be achieved without using actual sensors, but by injecting data directly to the IoT system.

2. Verify if the readings persisted within the IoT system are capable of triggering the expected alert;

3. Verify if all the subscribers to the alert(s) are notified.

   It is important to account for possible delays between the triggering of the alert(s) and the reception of the notification(s). The maximum acceptable delay will depend on the nature of the application and the technology used to produce the actual notifications (email, SMS, etc.).

Example:

Take again an IoT system which includes a blood pressure monitor, which can be used to diastolic and systolic pressure, as well as heart rate. Consider now that the system is configured to issue an alert when the blood pressure is either too high or too low, notifying a doctor who can then provide advice on how to proceed. This pattern makes it possible to ensure that the alert is issued when the blood pressure readings indicate a problem, and that the doctor is notified, by providing a set of inputs, which includes the test runtime, sensor settings — with regard to id and type of readings performed —, the means for accessing the data collected, and the details about the alert itself — with respect to the triggering condition(s), the notification and the subscribers.

3.2.4 Test Actions

Name:

Test Actions

Context:

Several IoT applications involve triggering the execution of some action whenever a certain condition is met. As with the case of alerts, these conditions are usually dependent on the data collected by the sensors.

Problem:

It must be ensured that a pre-set action is triggered and executed, under normal operation. Therefore, it is necessary to:

- Check that the expected action has been triggered;
- Check that the action was performed with a maximum delay, ensuring that the actuator's initial and final states are as expected.

Solution:

The test pattern for this type of behavior is only applied when an action is expected to be executed when a specific condition — or set of conditions — dependent on readings collected by one or more sensors is met. It consists of the following steps:

1. Generate readings capable of triggering the expected action(s);

   This can be attained either by using physical sensors and possibly actuators, to produce readings within certain thresholds. This would involve:

   (a) Setup the sensor(s);

   This may include specifying the type of readings to be performed.

   (b) Setup the actuator(s);

   This may include resetting the actuator to a particular state.

   (c) Setup a gateway (optional);

   This may include configuring it to maintain a log of the data collected, so that it can be compared with the data persisted.

   Alternatively, this could be achieved without using actual sensors, but by injecting data directly to the IoT system.

2. Verify if the readings persisted within the IoT system are capable of triggering the expected action(s);

3. Verify if the action was performed, within the pre-set maximum delay.

   It is important to account for possible delays between the triggering of the action and the actual execution. Verifying if the action was executed may be achieved by checking the state of an actuator or the readings performed by a sensor.

Example:

Take once more an IoT system which includes temperature and air quality sensors, which are setup to periodically perform measurements of the environment the system is in. Consider now that actuators are installed to control the opening and closing of window blinds given the average ambient temperature. Using this pattern it becomes possible to ensure that this pre-set action is triggered and executed, by providing a set of inputs which including the test runtime, sensor settings — with regard to id and type of readings performed —, the means for accessing the data collected, and the details about action itself — with respect to the triggering condition(s) and the expected final states.

3.2.5 Test Actuators

Name:

Test Actuators

Context:

Although IoT devices can be capable of contextual awareness, sensing capability, and some degree of autonomy, several IoT applications require the execution of actions upon command.

Problem:

It must be ensured that an actuator is capable of executing actions upon command, changing its state accordingly, under normal operation.

Therefore, it is necessary to:

- Check that the expected action has been triggered;
- Check that the action was performed with a maximum delay, ensuring that the actuator's initial and final states are as expected.

Solution:

The test pattern for this type of behavior is only applied when an action is expected to be executed. It consists of the following steps:

1. Setup the actuator(s);

   This may include resetting the actuator to a particular state.

2. Execute one or more commands;

3. Verify if the actions were performed.

   It is important to account for possible delays between the triggering of the action and the actual execution.

Example:

Take once more an IoT system which includes temperature and air quality sensors — setup to periodically perform measurements of the environment the system is in — and actuators that control the opening and closing of window blinds given the average ambient temperature. Consider now that it should be possible to issue commands that open or close those blinds. Using this pattern it becomes possible to ensure that those commands executed, by providing a set of inputs which include the test runtime, actuator settings — with regard to id and communication details — and a set of commands and expected states.

3.3 Summary

A feature model was devised so as to make it possible to categorize and group the components of an IoT ecosystem according to their characteristics. By enabling the representation of the plurality of components and features of IoT ecosystems, this allowed the identification of a set of recurring behaviors of IoT applications and a set of corresponding test strategies — which were described as IoT test patterns.

Chapter 4

Izinto Test Framework

4.1 Functionality 39
4.2 Architecture and Implementation 41
4.2.1 Test Logic Module 42
4.2.2 IoT Module 45
4.3 Usage 48
4.3.1 Test Duration 49
4.3.2 Data Sources 49
4.3.3 Devices 50
4.3.4 Triggers 51
4.4 Summary 53

This chapter describes *Izinto*, a pattern-based IoT testing framework that implements the patterns listed in Chapter 3, with respect to its functionality, architecture, implementation details, and usage.

4.1 Functionality

Named after the Zulu word for *things*, *Izinto* is a pattern-based IoT testing framework that aims to support the process of integration testing of IoT ecosystems. It makes it possible to test recurring behaviours in the scope of IoT in an automated manner and without the need for dealing with test logic, since it implements a set of IoT test patterns out-of-the-box. Furthermore, it reduces the effort that needs to be put into the configuration of communication protocols for the various components that make up the SUT, by supporting the most widely used technologies. As of now, the framework implements the patterns listed in Chapter 3, and its functionality is, therefore, closely related to those patterns. The framework makes it possible to:

- Test Periodic Readings

  Test IoT systems in which devices periodically perform measurements on one or more parameters, ensuring that those readings are performed at the expected rate and correctly transmitted and persisted within the IoT system.

- Test Triggered Readings

  Test IoT ecosystems in which devices perform measurements triggered by a user on one or more parameters, ensuring that those readings are correctly transmitted and persisted within the IoT system.

- Test Alerts

  Test scenarios which involve generating alerts whenever a certain condition is met, ensuring that the alert is triggered when that condition is met and that all subscribers are notified.

- Test Actions

  Test IoT applications involving the triggering and execution of some action whenever a certain condition dependent on the data collected is met, ensuring that a pre-set action is triggered and executed.

- Test Actuators

  Test IoT applications that require the execution of actions upon command, ensuring that an actuator is capable of executing those actions, changing its state accordingly.

To make this possible, it is necessary to specify the configuration of the IoT ecosystem, in terms of devices, and the triggering of alerts and actions.

Additionally, when relevant, it is possible to specify different data sources. A data source makes it possible to access readings collected within the SUT. These can be IoT components where the data collected is stored — such as gateway or databases —, or communication channels used to transmit those readings between the components of the IoT SUT — for instance, an MQTT broker.

Izinto also accounts for the possibility of manipulating the readings performed by the actual sensors (for instance, by using an actuator) or injecting readings directly into the SUT, so as to be able to trigger specific alerts and actions.

Currently, the framework supports the most widely used technologies with regard to devices and data sources — specifically, devices supporting MQTT communications, and MQTT brokers and REST APIs [Fie00] as data sources.

4.2 Architecture and Implementation

The framework was implemented in *Java 8* [Ora], and its architecture is depicted in Figure 4.1. In this architecture, it is possible to distinguish two main modules: one module which encompasses the actual test logic (package *TestHandlers*) and another module corresponding to abstractions of IoT components (other packages).

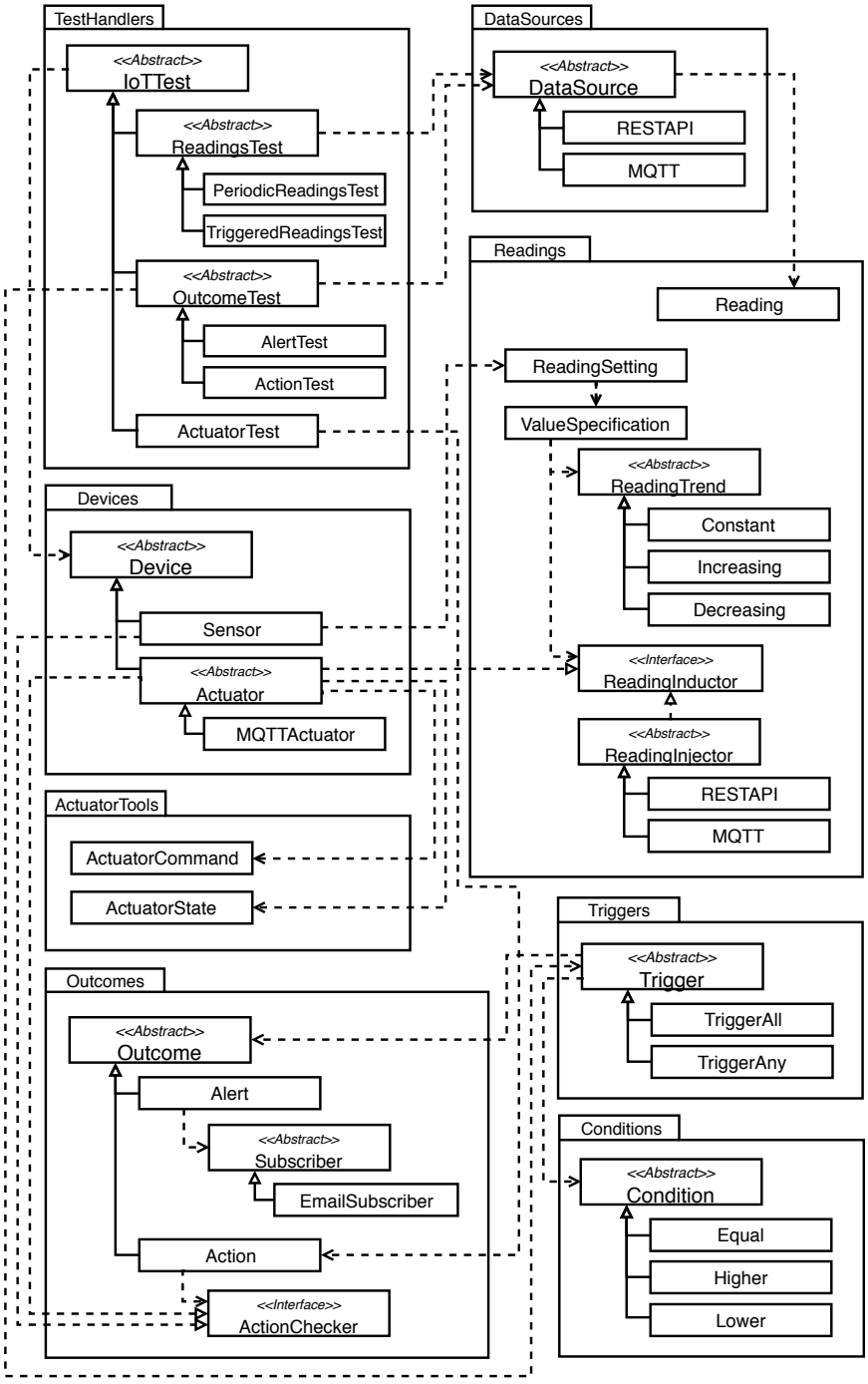


Figure 4.1: Representation of the framework's architecture.

4.2.1 Test Logic Module

The test logic was implemented using JUnit [JUn] — a UT framework for Java — and follows a set of IoT test patterns. Each check listed in the description of each pattern is performed in a different method with the JUnit *@Test* annotation, using JUnit's *Assertions*.

As of now, the framework implements the IoT test patterns described in Section 3.2, and each pattern is mapped into a specific class. Analyzing those patterns, it is possible to identify a set of checks that are common across more than one pattern. This allowed, from an architectural stand point, to establish a relation of inheritance between the logic of the several patterns, captured in Figure 4.2. As such, the logic behind each pattern can be spread out through more than one class, as detailed next.

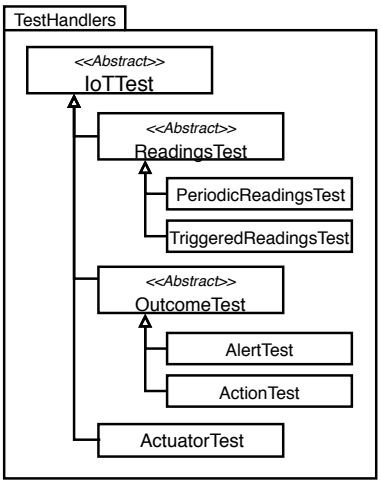


Figure 4.2: Package *TestHandlers*, corresponding to the module encompassing test logic.

IoTTest

IoTTest is an abstract class, which serves as the base implementation of all patterns.

To fully comprehend the test flow, it is important to understand the functioning of JUnit, illustrated in Figure 4.3. Methods with JUnit's *@BeforeClass* and *@AfterClass* annotations are static methods, being executed before the class is instantiated and after all the tests in the class have been run, respectively. Because they can only be used with static methods, it is impossible to perform setup operations based on the configuration file specified. Alternatively, methods with the *@Before* and *@After* annotations are executed before and after each method with the *@Test* annotation, and allow, for example, to reinitialize some attributes used in these methods and perform clean-up operations. However, such behaviour is not desirable in this situation, as it makes sense to perform the several checks associated with each test pattern on data obtained on the same test run. As such, the *setUp()* and *tearDown()* methods were annotated with *@Before* and *@After*, respectively, but were implemented in such a way that ensures that they are, in fact, executed only once, resembling the behaviour enforced by methods with JUnit's *@BeforeClass* and *@AfterClass* annotations.

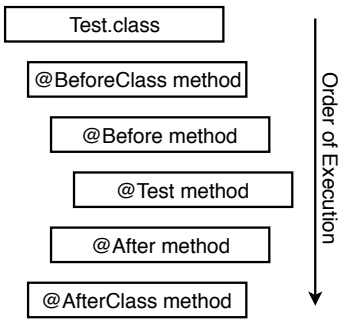


Figure 4.3: Order of execution of methods with different JUnit annotations.

Excerpt 4.1 presents part of the source code for *IoTTest* and provides some insight into the test flow. Next, the necessary setup steps are performed for each IoT component considered — including sensors, actuators, and data sources. With these components configured, the IoT system is allowed to function for the period specified as test runtime, before the checks pertinent to each IoT test pattern are performed. Finally, tear down and cleanup operations are performed for each device and data source configured.

```
1  IoTTest(JSONObject testConfiguration) {
2     parseDeviceSettings(testConfiguration);
3     parseDataSources(testConfiguration);
4  }
5
6  @Before
7  void setUp() {
8     if (!setUp) {
9         setUpDevices();
10        setUpDataSources();
11        setUp = true;
12    }
13 }
14
15 @After
16 void tearDown() {
17    if (!tornDown) {
18        tearDownDevices();
19        tearDownDataSources();
20        tornDown = true;
21    }
22 }
```

Excerpt 4.1: Excerpt of code reproducing the test flow in *IoTTest*.

ReadingsTest

This abstract class extends *IoTTest*. It overrides the implementation of the method *run()*, so as to — after having allowed the IoT system to function for the period specified as runtime — obtain the readings performed from the specified data sources.

The class implements a set of tests common to both the pattern for testing periodic readings and the pattern for triggered readings, specifically:

- *delayCheck()*

  Check that the readings are transmitted with a delay below a pre-set maximum;

- *dataConsistencyCheck()*

  Check that the readings are correctly transmitted;

- *valueSpecificationCheck()*

  Check that the readings are valid and, optionally, that variations are observed.

PeriodicReadingsTest

This class extends *ReadingsTest*, implementing an additional test:

- *periodicityCheck()*

  Check that the readings are performed with the pre-set periodicity, with acceptable deviations.

TriggeredReadingsTest

This class extends *ReadingsTest*, but does not implement any additional tests.

OutcomeTest

This abstract class also extends *IoTTest*. It overrides the implementation of the method *run()*, so as to — after having allowed the IoT system to function for the period specified as runtime — obtain the readings performed from the specified data sources.

It implements a set of tests common to both the pattern for testing alerts and the pattern for testing actions, specifically:

- *conditionCheck()*

  Check the conditions for triggering the alert are met and the alert have been met;

AlertsTest

This class extends *OutcomeTest*, implementing one additional test:

- *notificationCheck()*

  Check that a notification reaches all the subscribers, within an acceptable delay.

ActionsTest

This class extends *OutcomeTest*, implementing a set of additional tests:

- *actionDelayCheck(), initialStateCheck(), finalStateCheck()*

  Check that the action was performed with a maximum delay, ensuring that the actuator's initial and final states are as expected.

ActuatorsTest

This class extends *IoTTest*, implementing a set of additional tests:

- *actionDelayCheck(), statesCheck()*

  Check that each action was performed with a maximum delay, ensuring that the actuator's state changes as expected.

4.2.2 IoT Module

This module is composed of a set of connectors for the actual IoT components — sensors, actuators and data sources — corresponding to different categories of components identified in the feature model defined in Section 3.1. These allow to configure and control said components, along with a set of classes that represent concepts such as readings and actuator commands, along with alerts and actions.

Data Sources and Readings

A data source makes it possible to access readings collected within the SUT. These can be IoT components where the data collected is stored — such as gateway or databases —, or communication channels used to transmit those readings between the components of the IoT SUT — such as an MQTT broker.

As seen in excerpt 4.2, the abstract class *DataSource* declares three abstract methods which must be implemented in order to extend it, so as support for additional sources. The *setUp()* and *tearDown()* methods are executed before the actual test is run and afterwards, respectively, so as to perform setup and cleanup operations — for instance, configuring the device to a particular setting before the test is executed and restoring it to its original configuration afterward.

```
1 abstract void setUp();
2
3 abstract void tearDown();
4
5 abstract List<Reading> getReadings(Long startTimestamp,
6                                     Long endTimestamp,
7                                     List<Device> devices);
```

Excerpt 4.2: Abstract methods declared within the class *DataSource*.

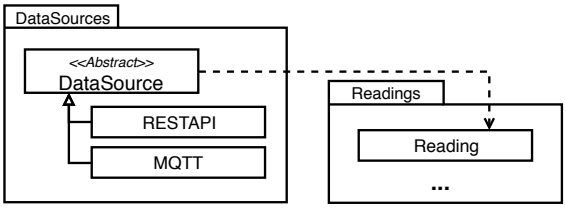


Figure 4.4: Relationship of dependency of *DataSources* on *Readings*.

The method *getReadings()* must retrieve the readings performed by the devices listed, during the period comprehended between the two timestamps provided. Figure 4.4 depicts this dependency on the class *Reading* — defined by five attributes (*type*, *sensor*, *timestampRead*, *timestampSaved*, and *value*) described on Table 4.1.

Attribute	Description
type	Description for the type of reading
sensor	ID of the sensor which performed the reading
timestampRead	Timestamp for the time the reading was performed by the sensor
timestampSaved	Timestamp for the time the reading was saved within the IoT system
value	Value of the reading

Table 4.1: Attributes for the class *Reading*.

Devices

A *Device* is identified by an ID, and is abstracted through adapters for the actual sensors and actuators. Figure 4.5 depicts the relation between these abstractions.

A *Sensor* extends the class *Device* and is further characterized by a list of reading settings — one per each different type of measurement that particular sensor supports. A particular reading type may be expected to be performed with a specific *interval*, as well as acceptable *deviation* and *delay* — which make for the attributes of a *ReadingSetting*.

To evaluate the correctness of the value of the readings performed, it is possible to define a *ValueSpecification*, which sets a maximum and minimum for the value of the readings, and may include a *ReadingTrend* — *Constant*, *Increasing*, or *Decreasing*.

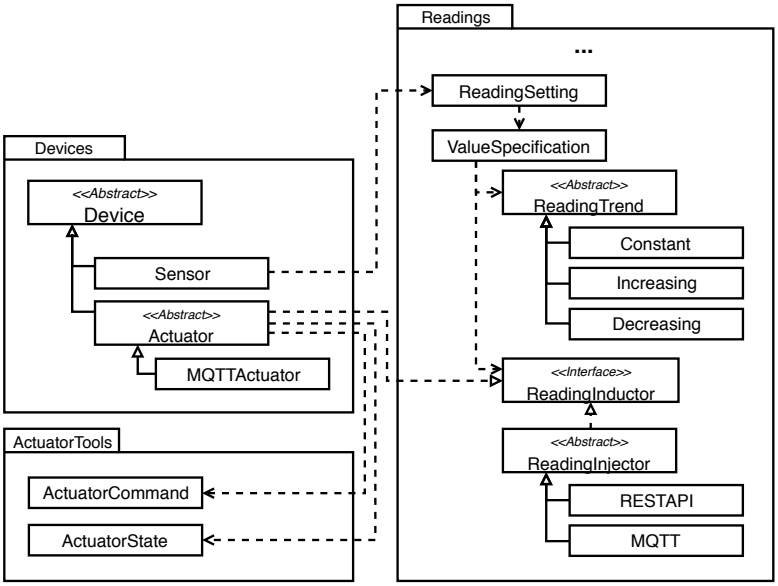


Figure 4.5: Relation between the classes related to devices and their configuration.

Furthermore, a *ReadingInductor* can be configured — this can be a physical device (i.e., an actuator which executes a set of commands that produce some variation over a sensor, affecting the readings performed) or a *ReadingInjector* (which will generate readings that match the corresponding value specification).

The class *Actuator* is an abstract class which extends the class *Device*, being further defined by a list of commands, through which its state can be changed. The way these commands are sent, necessarily, depend on the communication capabilities of the device itself.

Triggers, Conditions, and Outcomes

The class *Trigger* is defined by a set of conditions and outcomes. These outcomes can be triggered when any of the conditions specified is met (*TriggerAny*) or when all of them are met (*TriggerAll*). Figure 4.6 illustrates the relation between the classes related to triggers, conditions, and outcomes.

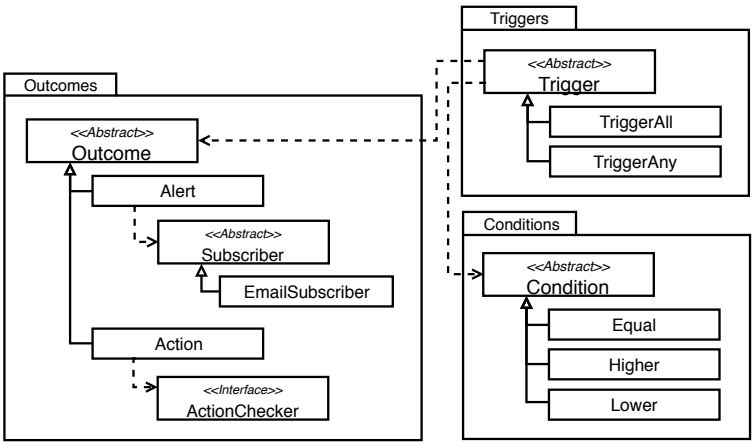


Figure 4.6: Relation between the classes related to triggers, conditions, and outcomes.

Conditions focus on the value of readings — of a specific type, being also possible to specify the ID for a particular sensor — and whether this value is equal to, or higher or lower than the reference value defined — corresponding to different types of conditions (*Equal*, *Higher*, *Lower*).

The abstract class *Outcome* has a description, and two timestamps — *triggerTimestamp* and *outcomeTimestamp* — allowing to check if the outcome was achieved within an *acceptableDelay*. Both the class *Alert* and *Action* extend this class, and implement the abstract method *checkOutcome()*, so as to set the *outcomeTimestamp* corresponding to the time of notification — in the case of an alert —, and the time of the change of state — in the case of an action.

4.3 Usage

Izinto was conceived aiming at two different profiles. On the one hand, it can be used by testing experts to put virtually any IoT solution to the test, as the framework can be extended so as to meet a particular need. On the other hand, those with minimal technical knowledge about testing can still use the framework, as only some knowledge about the SUT is required to configure it.

To use *Izinto* to test an IoT system, a configuration file must be specified, with the appropriate settings for the SUT. These specifications will depend on the tests which are to be performed and must include the settings for the devices which make up the SUT, as well as the settings for the data sources considered. To test actions and alerts, information on the triggers and expected outcomes must also be provided.

```
{
 "testDuration": <INTEGER>,
 "dataSources": [{
    "type": <STRING>,
    "specs": <JSON_OBJECT>
 }, ...],
 "devices": [{
    "type": <STRING>,
    "specs": <JSON_OBJECT>
 }, ...],
 "triggers": [{
    "type": <STRING>,
    "specs": <JSON_OBJECT>
 }, ...]
}
```

Excerpt 4.3: Possible settings within the configuration file.

As shown in excerpt 4.3, the configuration file includes four key settings — the test's duration, the settings for data sources and devices, and details on the alerts and actions to be tested — which are further explained next.

4.3.1 Test Duration

Test runtime — in milliseconds —, corresponding to the period of time considered for test purposes.

4.3.2 Data Sources

List of data sources that allow access to readings collected within the SUT. These can be IoT components where the data collected is stored — accessible via a REST API —, or communication brokers used to transmit those readings between the components of the IoT SUT — such as an MQTT broker —, and their configuration will depend on its type, as displayed in excerpt 4.4.

This list can include any number of data sources, depending on the SUT and the patterns that may be applied to it. For instance, in the cases in which a pattern for testing readings can apply, at least two data sources should be specified, so as to enable the check of consistency across the IoT system. Conversely, this list can be empty in the cases when the only applicable test pattern is that of the test of actuators, in which readings do not play a part. To test actions and alerts, information on the at least one data source should be provided, so as to check if the condition specified in the trigger was met.

```
{...,
 "dataSources": [{
    "type": "MQTTBroker",
    "specs": {
        "mqttBrokerAddress": <STRING>,
        "topics": [<STRING>, ...]
    }
 },
 {
    "type": "REST_API",
    "specs": {
        "url": <STRING>,
        "headers": <JSON_OBJECT>,
        "parameters": <JSON_OBJECT>
    }
 }, ...],
 ...
}
```

Excerpt 4.4: Settings for different data sources.

4.3.3 Devices

List of devices within the SUT and corresponding settings. Devices are identified by an ID and are abstracted through adapters for the actual sensors and actuators, allowing their configuration.

Sensors are characterized by a list of reading settings — one per each different type of measurement a particular sensor supports. These settings include a description for the type of reading, the expected interval — when the readings are expected to be performed periodically —, as well as the acceptable deviation and delay. Furthermore, it is possible to define a *valueSpecification*, setting a maximum and minimum for the value of the readings, and may include a *readingTrend* — *Constant*, *Increasing*, or *Decreasing*. A *readingInductor* can also be configured. This can be an actuator — which executes a set of commands that produce some variation over a sensor, affecting the readings performed —, or a *readingInjector* — which will generate readings that match the corresponding value specification.

```
{...,
 "devices": [{
    "type": "Sensor",
    "specs": {
        "id": <STRING>,
        "readingSettings": [{
            "type": <STRING>,
            "expectedInterval": <INTEGER>,
            "acceptableDeviation": <INTEGER>,
            "acceptableDelay": <INTEGER>,
            "valueSpecification": {
                "minimumValue": <DOUBLE>,
                "maximumValue": <DOUBLE>,
                "readingTrend": {
                    "type": <"Increasing"|"Decreasing"|"Constant">,
                    "specs": <JSON_OBJECT>
                },
                "readingInductor": <STRING|JSON_OBJECT>
            }
        }, ...]
    }
 }, ...],
 ...
}
```

Excerpt 4.5: Settings for sensors.

In the case of actuators, it is possible to control the devices, through commands. An MQTT actuator requires the definition of an MQTT broker and the MQTT topic associated with the actuator's state. It is possible to specify a setup command, to be executed before the actual test process begins. The set of commands provided are then executed sequentially with specific intervals — specified in milliseconds.

```
{...,
 "devices": [{
    "type": "MQTTActuator",
    "specs":{
        "id": <STRING>,
        "setUpCommand": {
            "message": <STRING>,
            "command": <STRING>,
            "args": <STRING>
        },
        "commands": [{
            "message": <STRING>,
            "interval": <INTEGER>,
            "command": <STRING>,
            "args": <STRING>
        }, ...],
        "mqttBrokerAddress": <STRING>,
        "stateTopic": <STRING>
    }
 }, ...],
 ...
}
```

Excerpt 4.6: Settings for actuators.

4.3.4 Triggers

List of triggers defining conditions for the triggering of alerts and actions. These alerts and actions can be triggered when any of the conditions specified is met (*TriggerAny*) or when all of them are met (*TriggerAll*). These *conditions* focus on the value of readings — of a specific *type*, being also possible to specify the ID for a particular *sensor* — and whether this value is *equal* to, or *higher* or *lower* than the reference value defined — corresponding to different types of conditions. The creation of more flexible trigger and condition types can be considered as future work (see Chapter 6).

In turn, the list of *outcomes* can include alerts and actions. In both cases, a *description* must be provided, along with the *acceptable delay* (in milliseconds) between the triggering of the condition and the time of notification or execution of the action. In the case of an alert, it is also necessary to specify a list of *subscribers*, i.e., those who must be notified when the alert is triggered, so as

to verify if the notification was received. The settings for these subscribers depends on the type of notification, as, as of now, the framework supports email notifications. In the case of an action, it is also required to specify the expected *initial* and *final states*, along with an *action checker* — which can be either a sensor or the actuator performing that action —, through which the initial and final states can be verified.

```
{...,
 "triggers": [{
    "type": <"TriggerAll"|"TriggerAny">,
    "specs": {
        "conditions": [{
            "type": <"Higher"|"Equal"|"Lower">,
            "specs": {
                "sensor": <STRING>,
                "type": <STRING>,
                "value": <STRING>
            }
        }, ...],
        "outcomes": [{
            "type": "Alert",
            "specs": {
                "description": <STRING>,
                "acceptableDelay": <INTEGER>,
                "subscribers": [{
                    "type": "EmailSubscriber",
                    "specs": {
                        "email": <STRING>,
                        "password": <STRING>
                    }
                }, ...]
            }
        },
        {
            "type": "Action",
            "specs": {
                "description": <STRING>,
                "acceptableDelay": <INTEGER>,
                "expectedInitialState": <STRING>,
                "expectedFinalState": <STRING>,
                "actionChecker": <STRING|JSON_OBJECT>
        },...]
    }
 }, ...],
 ...
}
```

Excerpt 4.7: Settings for different triggers, conditions, and outcomes.

4.4 Summary

The framework developed was described with respect to its functionality, closely related to the IoT test patterns it implements out-of-the-box. *Izinto*'s modular architecture was analyzed and implementation details were provided. Having been developed as a modular framework, it becomes simple to extend it, with respect either to the test logic — by creating other test patterns that might emerge —, or to the abstractions of IoT components — by adding support for other technologies, namely new types of devices and/or data sources. Requiring only a configuration file for it to work, it can be used by testing experts and by those with minimal technical knowledge about testing as well. These specifications include the settings for the devices which make up the SUT, as well as the settings for the data sources considered. To test actions and alerts, information on the triggers and expected outcomes must also be provided.

Chapter 5

Validation

5.1 Application Scenario 56
5.1.1 Functionality 57
5.1.2 Components 59
5.1.3 Architecture and Implementation 62
5.2 Test cases 67
5.2.1 Temperature/Humidity Test 70
5.2.2 Heart Rate Monitoring Test 71
5.2.3 Air Quality Monitoring Test 72
5.2.4 Luminosity Test 73
5.2.5 Weight Monitoring Test 74
5.2.6 Blood Pressure Monitoring Test 75
5.3 Results 76
5.4 Discussion 77
5.5 Summary 78

This chapter describes the validation of the framework proposed and described in Chapter 4, using a concrete IoT application scenario in the domain of AAL.

5.1 Application Scenario

As demonstrated in Chapter 1, healthcare is one of the domains which can benefit of IoT. For instance, in the case of elderly or chronically ill patients, RPM can provide valuable information in the home and make it possible to avoid prolonged hospital stays or reduce the number of visits to medical facilities. With the help of RPM systems, health parameters can be accessed by doctors or other care givers, even when they are not in the vicinity. Thus, the scenario considered corresponds to a remote health monitoring application consistent with an application in the domain of AAL. It is depicted in Figure 5.1 and involves two actors: a care receiver — the subject being monitored — and a care giver — the person monitoring the patient.

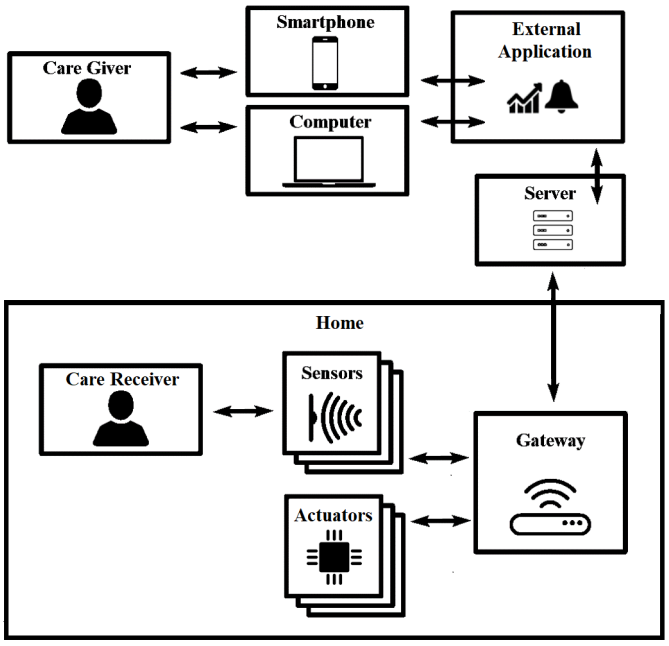


Figure 5.1: Architecture of the application scenario.

Various sensors collect and transmit data both on the patient's health parameters and on ambient conditions to a server. This server maintains a personal health record and generates alerts if any of these parameters are determined to require attention, notifying a care giver and/or triggering actions through actuators.

5.1.1 Functionality

In accordance to the scenario described, the system implements the following functionality:

- Monitoring and maintaining a personal record of health parameters, namely heart rate, blood pressure, and weight;
- Monitoring ambient conditions, specifically temperature, humidity, air quality and luminosity;
- Issuing alerts when the parameters being monitored indicate a problem:
  - elevated heart rate;

    The alerts generated reflect the heart rate intervals defined in Table 5.1. The actual intervals are calculated — in beats per minute (BPM) — using an estimated maximum heart rate (MHR), following the stipulated by the American Heart Association (AHA) [Ass18b].

Category	Interval
Normal	<50%
Elevated	51% - 69%
High	70% - 84%
Very High	>85%

Table 5.1: Categorization of heart rate for different intervals, defined as a percentage of the MHR.

  - dangerous blood pressure;

    Various alerts were set for blood pressure values considered to be abnormal, according to the intervals defined by the AHA 5.2 (see Table 5.2).

Category	Systolic mmHg (upper value)		Diastolic mmHg (lower value)
Normal	<120	and	<80
Elevated	120 - 129	and	<80
High (Hypertension, Stage 1)	130 - 139	or	80 - 89
High (Hypertension, Stage 2)	$\geq$ 140	or	$\geq$ 90
Hypertensive Crisis	$>$ 180	and/or	$>$ 120

Table 5.2: Healthy and unhealthy blood pressure ranges, as defined by the AHA [Ass18a].

– increase or decrease in weight to a value above or below pre-set thresholds (see Table 5.3);

Parameter	Threshold	
	Lower	Upper
Weight (Kg)	70	80

Table 5.3: Weight thresholds considered for the triggering of alerts.

– ambient temperature, humidity, air quality or luminosity rising or dropping below pre-set thresholds (see Table 5.4).

Parameter	Threshold	
	Lower	Upper
Temperature (°C)	15	25
Humidity (%)	30	70
Air Quality (ppm)	N/A	1000
Luminosity (Lum)	300	N/A

Table 5.4: Ambient condition parameters considered in the scenario and corresponding thresholds for the triggering of alerts.

- Triggering of actions upon certain conditions are met:
  - Turning on the lights when the luminosity drops below a pre-set threshold;
  - Turning on the AC when the ambient temperature drops rises above pre-set thresholds.

  The conditions considered for these actions are described in Table 5.5.

Parameter	Condition	Action
Temperature	> 25	Turn on AC
	< 22	Turn off AC
Luminosity	< 300	Turn on lights

Table 5.5: Triggering conditions and respective actions considered in the scenario.

- Consult a personal health record and the record of ambient conditions via a web application.

5.1.2 Components

In this application scenario, the system was composed of several sensors — which can generically be categorized as body or ambient sensors — and actuators. It must also be considered the existence of a gateway device which will collect the data from the various sensors — and transmit it to a server — and will trigger pre-determined actions according to the condition of the patient or the surrounding environment. The server maintains a record of the data collected — which can be accessed through any computer or smartphone via a web application — and generates alerts if some of the parameters monitored is determined to require attention, notifying a care giver so that it can be determined if additional measures must be taken. Figure 5.2 portrays these components.

(a) Fitness wristband

(b) Weight Scale

(c) Blood pressure monitor

(d) Temperature and humidity sensor

(e) Air quality sensor

(f) Light dependent resistor

(g) Light bulb holder

(h) Smart socket

(i) Gateway

(j) Server

(k) Computer

(l) Smartphone

Figure 5.2: Main components used in the application scenario.

Body sensors

- Fitbit Charge 2

  The Fitbit Charge 2 — shown in Figure 5.2a is a wearable device, specifically a heart rate and fitness wristband that allows the tracking of exercise, sleep, and heart rate [Fitb]. The data collected can be automatically synchronized with a mobile phone, via Bluetooth, using the Fitbit app [Fita], which will in turn upload this data to Fitbit's dashboard.

  In this scenario, the feature to be explored is the tracking of resting heart rate of the subject being monitored over time. This data is accessible via Fitbit's Web API [Fitc].

- Nokia Body+ Scale and Nokia BPM

  The Nokia Body+ Scale — shown in Figure 5.2b — is a weight scale with the ability to perform full body composition analysis, measuring weight, body fat and water percentage, as well as muscle and bone mass [Noka]. It can be setup to automatically synchronize measurements with a mobile phone, via Bluetooth, and with Nokia's health dashboard, via Wi-Fi. There are mobile applications available for Android and iOS, which give users the ability to monitor their progress; the Nokia Health Mate [Nokc] app also tracks activity, weight, nutrition, sleep, and heart rate.

  In turn, the Nokia BPM — shown in Figure 5.2c — is a blood pressure monitor designed to measure human blood pressure — in terms of systolic and diastolic pressure —, as well as pulse rate, and has been certified by the Food and Drug Administration, in the United States. To perform a measurement, this monitor must be used in conjugation with the Health Mate app, which launches automatically upon turning on the device. Data from every measurement is automatically synchronized with the app via Bluetooth, that will also synchronize it with Nokia's health dashboard and provide insights based on internationally recognized standards.

  The data collected using these devices can be accessed by third-party applications via the Web API [Nokb] provided by Nokia Health.

Ambient Sensors

- AM2302 Temperature/Humidity Sensor

  The AM2302 sensor — shown in Figure 5.2d — is a basic digital temperature and humidity sensor and consists of a capacitive humidity sensor and a thermistor to measure the surrounding air [DK]. Some important specifications include its accuracy ($\pm 2\%$) and response time (2 seconds). Additionally, this sensor allows to obtain temperature readings ranging from -40 to +80 degrees Celsius and humidity between 0 to 100% and is commonly used with Arduino, Raspberry Pi, and other micro-controllers.

- GL5528 LDR Sensor

  The Light Dependent Resistor (LDR) — shown in Figure 5.2f — is a component whose resistance varies according to the light intensity. Since the resistance offered by the LDR decreases as the level of luminosity increases in a predictable way — 1 $M\Omega$ for 0 Lux, and 10-20 $K\Omega$ for 10 Lux [Ele] — it is possible to measure the resistance of the LDR sensor and thus obtain an estimation of the level of luminosity in the surrounding environment [Teca].

  This LDR light sensor is often used with Arduino as well as other micro-controllers, in applications such as alarms and home automation.

- MQ-135 Air Quality Sensor

  Also known as MQ-135 Air Quality Sensor, the MQ-135 gas sensor — shown in Figure 5.2e — is a module capable of detecting various types of toxic gases such as Ammonia, Carbon Dioxide, Benzene, and Nitric Oxide, as well as smoke and alcohol [PTRc].

  This sensor — in conjunction with a micro-controller such as Arduino —, allows the setup of monitoring and alarm systems based on ambient gas concentration.

Actuators

- Sonoff Slampher and Sonoff S20

  The Sonoff Slampher — shown in Figure 5.2g — is a WiFi smart light bulb socket that can be connected to light bulbs, enabling users to remotely control connected light bulbs [ITE16b]. In turn, the Sonoff S20 — shown in Figure 5.2h — is a WiFi smart plug that allows users to convert any plug into a smart outlet [ITE16a].

  Out-of-the-box, it is possible to control these devices through the mobile application *eWeLink* — available for iOS and Android —, which allows users to control the devices by turning them on/off from anywhere at any time, or to set scheduled timers to automate this.

Gateway

- Raspberry Pi 3 Model B

  The Raspberry Pi 3 Model B — shown in Figure 5.2i — is a small and low cost device which works in a similar fashion as a standard PC, requiring a keyboard for giving commands, a display unit and power supply [FOUa]. The board itself contains a Quad Core 1.2GHz Broadcom BCM2837 processor, 1GB RAM memory, and connectors and interfaces to other devices. The Raspberry Pi has a large number of applications. Having support for Ethernet, WiFi, and Bluetooth communication, it is an extremely well suited platform for interfacing with several devices and — for this reason — was selected to act as a gateway.

Server

- Virtual machine

  The virtual machine used Linux Ubuntu 16.04.3 LTS, running on a machine with a Intel(R) Core(TM) i7-7700K CPU@4.20GHz processor and 4GB of RAM — as the one shown in Figure 5.2j — acting as the system's server. Details on its configuration are provided later on.

Other components

- Espressif ESP32

  The Espressif ESP32 is a high-performance module, with 4MB of flash memory and low power consumption, which comes with WiFi and BLE built-in and can be programmed via micro-USB [PTRa].

- Computer and Smartphone

  A computer and a smartphone — shown in Figures 5.2k and 5.2l, respectively — were used for monitoring health and ambient parameters through a web application, as well as to receive notifications. The smartphone served an additional purpose, acting as an intermediary between the body sensors and the corresponding proprietary APIs, as explained farther on.

- Circuit Components
  - Breadboard
  - Jumper wires
  - Resistor (100 Ω)

5.1.3 Architecture and Implementation

The development of this scenario started with the definition of use cases, which reflected the functionality described. This being an IoT system, it fits the paradigm in which different components act as sensors, actuators, or processors and communicate with each other to implement the system's functionality. In this scenario, the several sensors communicate with a gateway which ensures the connection to a server, responsible for persisting the data collected by the various sensors, generating alerts based on this data, and hosting a web application which provides access to this data.

Next, the technology to be used was selected, with respect to hardware and infrastructure, IoT platform or middleware, and visualization tools. The implementation of the scenario consisted in a work of integration, taking advantage of existing middleware solutions.

Body Sensors

The body sensors considered — Nokia BPM, Nokia Body+, and Fitbit — are proprietary, and do not have open Software Development Kits (SDKs). As such, the system's gateway is unable to establish communication with these devices directly. Instead, they communicate with a smartphone via Bluetooth, and transmit the readings performed to the respective servers — Nokia Health and Fitbit — through the corresponding proprietary mobile application — Health Mate and Fitibit App. The system's gateway retrieves the readings performed by these devices through the REST APIs provided by Nokia Health and Fitbit. This behaviour is captured in Figure 5.3.

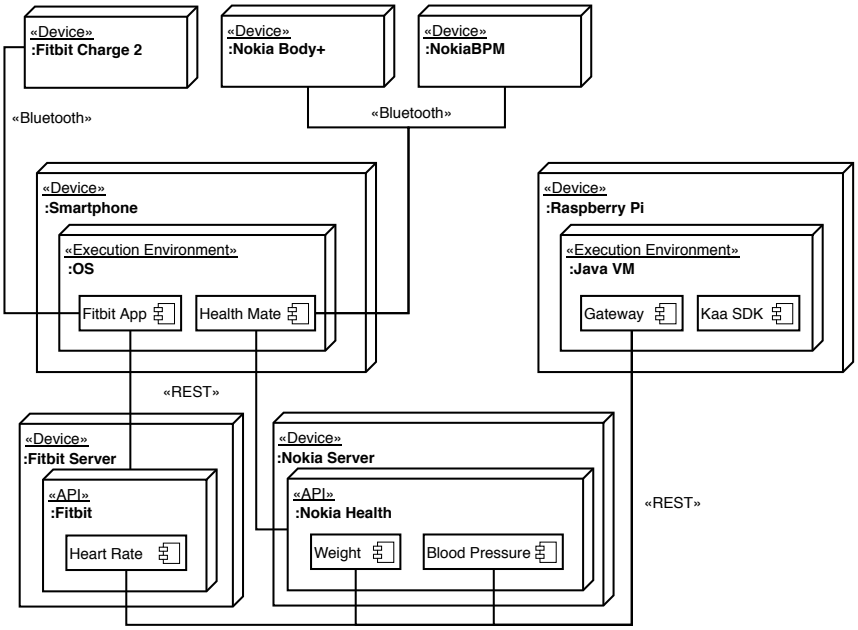


Figure 5.3: Deployment view depicting the connection between the body sensors and the system's gateway.

Ambient Sensors

The ambient sensors listed are very low level devices, devoid of processing and communication capabilities. Because of that, the various ambient sensors — AM-2302, MQ-135, and LDR sensor — were connected to the ESP32 board via its input/output (I/O) pins. Being a cross-platform code builder and library manager, PlatformIO IDE was used in the development of the software for interfacing with these lower level devices and to flash the ESP32 module with it. Figure 5.4 corresponds to a schematic representation of the assembling of these sensors and the module ESP32.

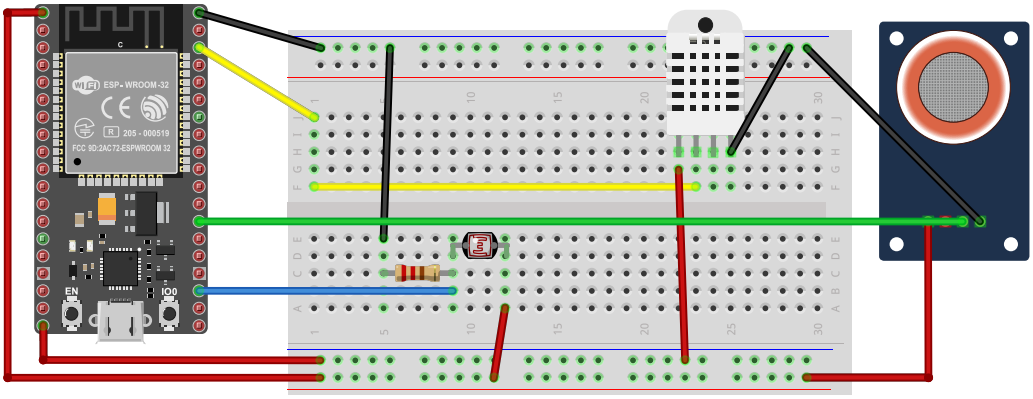


Figure 5.4: Assembling of the ambient sensors and the ESP32 module.

The readings of the AM2302 Temperature/Humidity sensor were obtained using the Adafruit DHT library [Ind], an Arduino library for the DHT series of Temperature/Humidity sensors, which provides simple methods to retrieve temperature and humidity measurements from the sensor.

The LDR Sensor is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity, making it possible — after some experimentation and calibration — to estimate the level of luminosity. It is important to note that this is only an estimate and does not provide an accurate level of luminosity, but it is sufficiently accurate to to distinguish dark from light environments.

In turn, the readings from the MQ-135 Air Quality Sensor were obtained using a library [PTRc] that handles the retrieval of readings from the sensor corresponding to the amount of CO_2 measured in PPM.

The readings from these sensors are retrieved and sent from the ESP32 module to the gateway using the MQTT protocol. This architecture is illustrated in Figure 5.5.

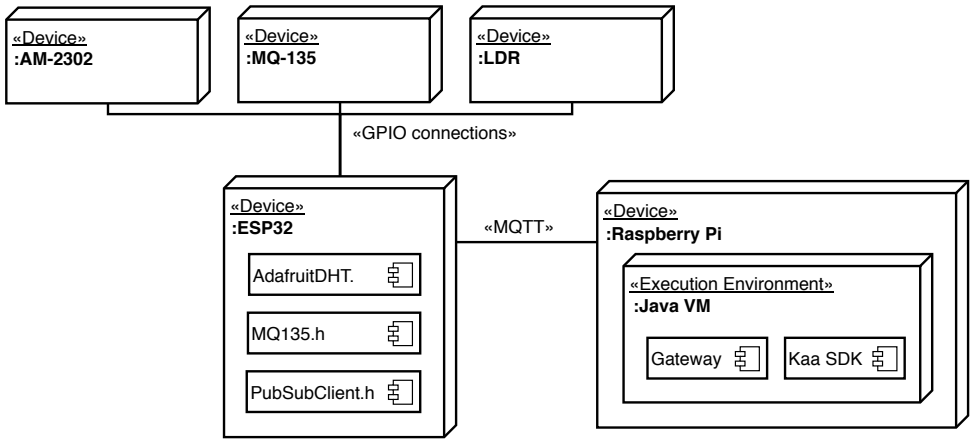


Figure 5.5: Deployment view depicting the connection between the ambient sensors and the system's gateway.

Actuators

The actuators considered come out-of-the-box with proprietary firmware and can only be controlled through a proprietary mobile application. To overcome this limitation and enable the triggering of actions by the gateway — as illustrated in Figure 5.6 —, an alternative firmware, Tasmota [Are], was uploaded to the devices through PlatformIO, using a 3.3*V* FTDI module [PTRb]. Tasmota is a firmware for ESP8266 based devices — as it is the case of the Sonoff Slampher and the Sonnof S20 —, allowing them to be controlled over HTTP and MQTT.

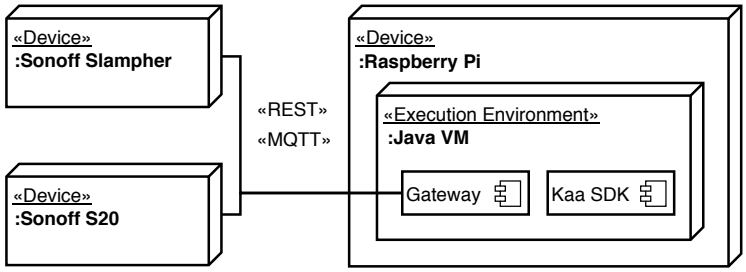


Figure 5.6: Deployment view depicting the connection between the actuators and the system's gateway.

Gateway and Server

With respect to middleware, Kaa was selected to handle data collection and persistence. It was installed in the server and configured so as to receive data from the gateway. An MQTT broker was also setup, so as to allow MQTT communications. The server also hosts a dashboard — supported by Redash [Red] — through which the readings collected could be visualized — in real time (Figure 5.7) and historically (see Figure 5.8) — and the alerts managed (Figure 5.9).

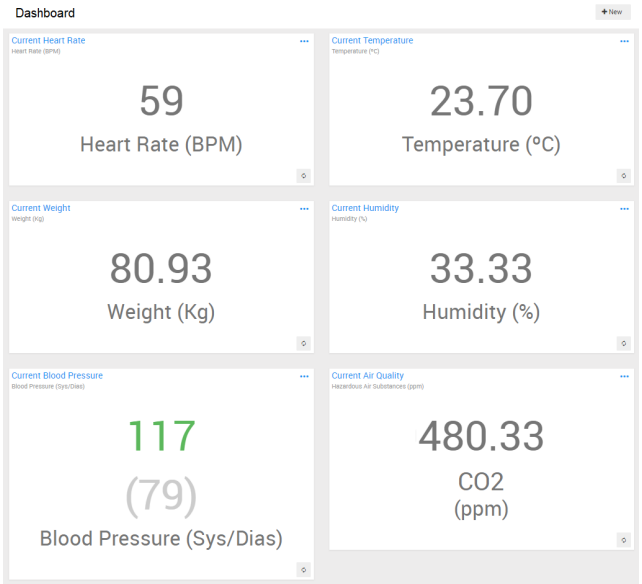


Figure 5.7: Dashboard's widgets for the visualization of real time data.

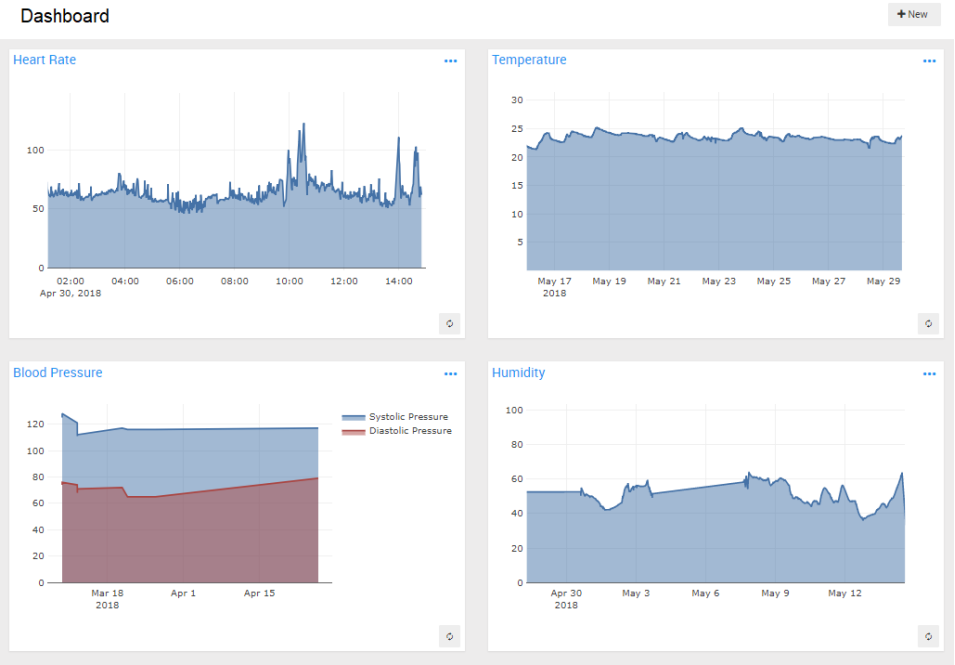


Figure 5.8: Dashboard's widgets the visualization of historic data.

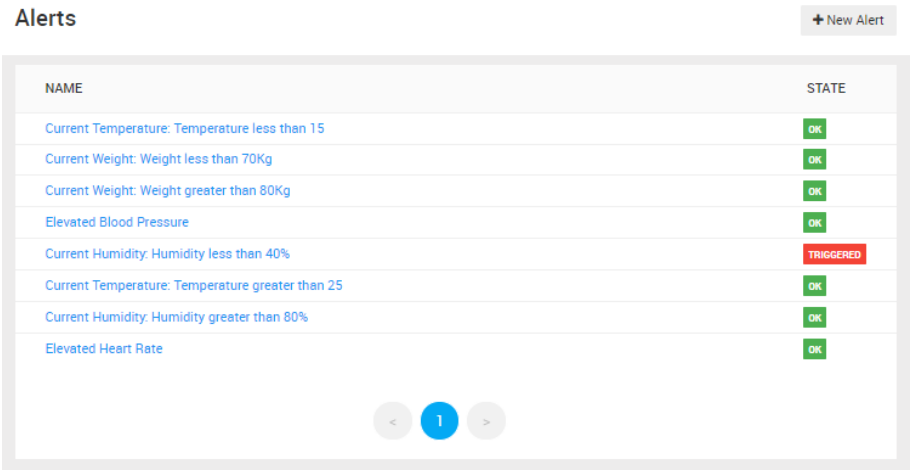


Figure 5.9: Dashboard's interface for alert management.

The *Gateway* is a minimal Java application deployed on the Raspberry Pi, and configured so as to collect readings from the several sensors and trigger actions based on those readings. A Kaa SDK [Kaa] made it possible to send the data collected to the server in a seamless manner, as depicted in Figure 5.10.

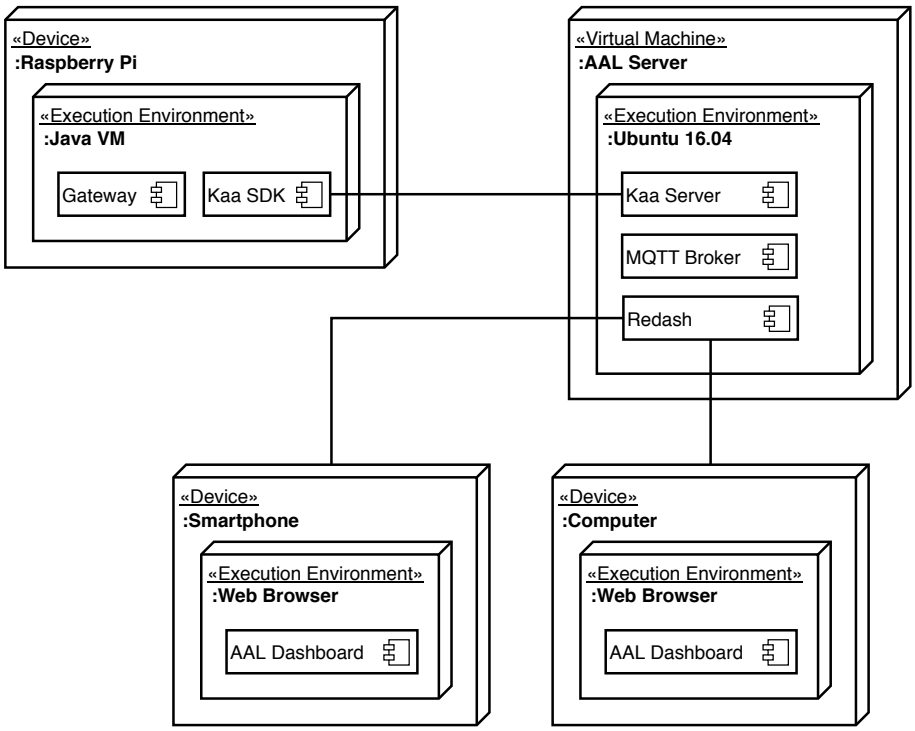


Figure 5.10: Deployment view depicting the connection between the system's gateway and the system's server.

5.2 Test cases

Within the application scenario described, it was possible to instantiate different test patterns in different test cases. The test process used for validation purposes consisted in three distinct phases:

- **Analysis**

  Characterization of the components of the SUT, in accordance to the feature model proposed in Section 3.1;

- **Design**

  Mapping the features to applicable test patterns;

- **Implementation**

  Test configuration, through the specification of the SUT's settings.

Table 5.6 presents the characterization of the components of the SUT, in accordance to their features. It should be noted that this characterization is solely based on the features considered in the scenario and not on all the features the components offer.

Device	Category	Features
Fitbit Charge 2	Sensor	Periodic Readings, Single Measurement
MQ-135	Sensor	Periodic Readings, Single Measurement
LDR	Sensor	Periodic Readings, Single Measurement
AM-2302	Sensor	Periodic Readings, Multiple Measurements
Nokia Body+ Scale	Sensor	Triggered Readings, Multiple Measurements
Nokia BPM	Sensor	Triggered Readings, Multiple Measurements
Sonoff Slampher	Actuator	N/A
Sonoff S20 Socket	Actuator	N/A

Table 5.6: Devices and corresponding features.

Having identified the characteristics of the various system components, it was possible to identify a set test cases, in which was possible to instantiate different test patterns.

Take as an example the case in which a AM-2302 sensor is capable of transmitting periodic readings of multiple types — Temperature and Humidity —; a gateway collects, processes and transmits those readings to a server — which handles communications and data management — and controls an air conditioner (AC) through an actuator, based on the condition $Temperature > 25$. Additionally, an external application generates alerts when the humidity drops below 30% or rises 70%.

```
{"type": "Sensor",
 "specs": {"id": "AM-2302", "readingSettings":
          [{"type": "Temperature",
            "expectedInterval": 60000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": -40,
                                   "maximumValue": 80}},
           {"type": "Humidity",
            "expectedInterval": 120000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": 0,
                                   "maximumValue": 100}}]
}}
```

Excerpt 5.1: Configuration for testing periodic temperature and humidity readings.

The test configuration phase corresponded to the specification of the SUT's settings through a configuration file, which follows the guidelines outlined in Section 4.3. In this case, Excerpt 5.1 corresponds to part of the configuration file required for testing the AM-2302 sensor, expected to perform temperature readings every minute and humidity readings every two minutes, with a maximum deviation of 3 seconds and a maximum transmission delay of 2 seconds. As mentioned in subsection 5.1.2, the sensor's specification states it is capable of measuring temperatures from -40ºC up to 80ºC and that it measures the relative humidity from 0% up to 100%.

```
{"type": "TriggerAll",
 "specs": {"conditions": [{"type": "Higher",
           "specs": {"sensorID": "AM2302",
                     "type": "Temperature",
                     "value": 25
           }}],
           "outcomes": [{"type": "Alert",
           "specs": {"description": "Temperature > 25",
                     "acceptableDelay": 10000,
                     "subscribers": [{
                           "type": "EmailSubscriber",
                           "specs": {
                           "email": "kaahealth@gmail.com"
                     }}]
           }}]
}}
```

Excerpt 5.2: Configuration for testing a temperature alert.

In turn, Excerpt 5.2 corresponds to the part of the configuration file required for testing the triggering of an alert when the temperatures rises above 25ºC, which must be sent once via email and this notification must reach each subscriber within 10 seconds.

These test cases can be described in a more structured manner:

- **Identifier:** unique identifier to shortly refer the test case;
- **Name:** name to designate the test case;
- **Description:** description of the test case, with respect to the components and functionalities under test;
- **Test Patterns:** test patterns applicable to the test case, fitting the components and functionalities under test.

Next, the various test cases considered are described next following the template presented. A complete configuration file with the settings required to perform all theses cases is attached in full as Appendix A.

5.2.1 Temperature/Humidity Test

Identifier:

TC_01

Name:

Temperature/Humidity Test

Description:

This test case corresponds to the test case previously taken as an example in which the AM-2302 sensor performs, periodically, multiple readings — temperature and humidity measurements — which can be monitored via a dashboard. Temperature measurements above a pre-set level trigger the activation of the AC — via an actuator (the Sonoff S20 Socket) — and the measurements below or above pre-set levels trigger an alert. Figure 5.11 and shows the different IoT components inserted into the respective categories and characterized by their features.

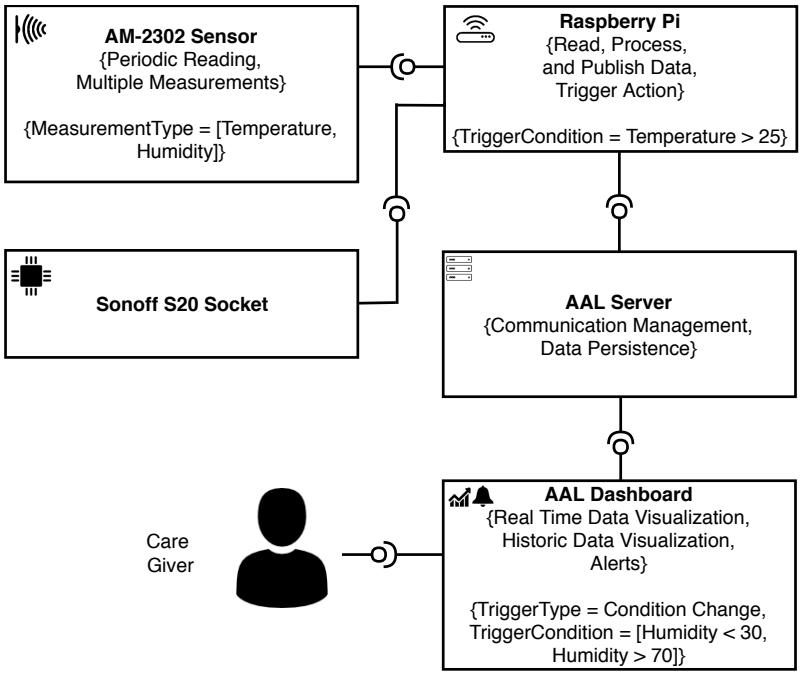


Figure 5.11: Architecture illustrating the test case for temperature/humidity features.

Test Patterns:

This test case involves testing the triggering of an alert and action following the reaching of a pre-set condition, and requires instantiating the following test patterns:

- Test Periodic Readings;
- Test Alerts;
- Test Actuators;
- Test Actions.

5.2.2 Heart Rate Monitoring Test

Identifier:

TC_02

Name:

Heart Rate Monitoring Test

Description:

This test case considers that heart rate measurements are performed periodically by Fitbit Charge 2. These measurements can be monitored via a dashboard and an alert is generated when they indicate a problem. Figure 5.12 shows the different IoT components inserted into the respective categories and characterized by their features, allowing the instantiation of several test patterns.

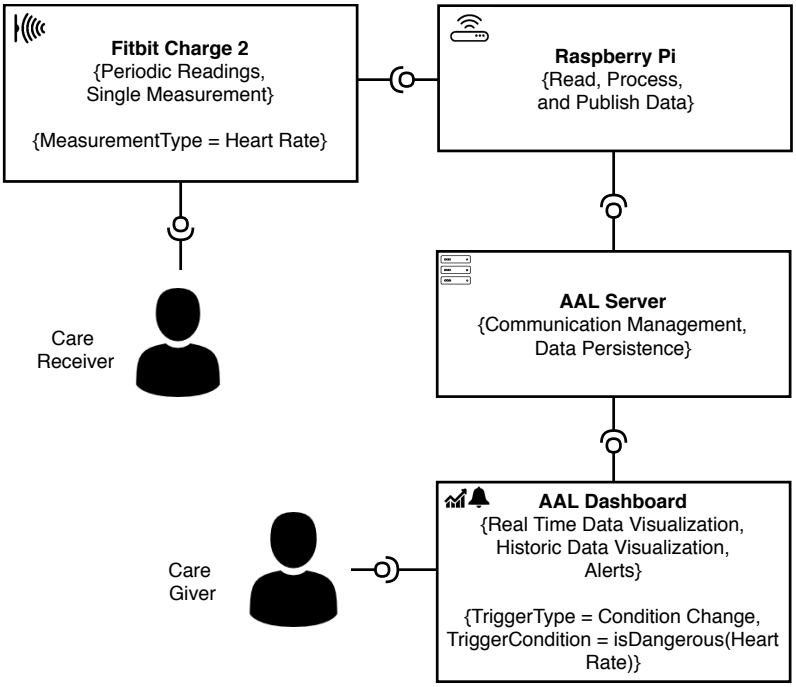


Figure 5.12: Architecture illustrating the test case for heart rate monitoring features.

Test Patterns:

This test case involves testing the triggering of an alert following the reaching of a pre-set condition, and requires instantiating the following test patterns:

- Test Periodic Readings;
- Test Alerts.

5.2.3 Air Quality Monitoring Test

Identifier:

TC_03

Name:

Air Quality Monitoring Test

Description:

This test case considers that the readings from the MQ-135 sensor are periodically collected as air quality measurements, and that when these measurements indicate a problem an alert is triggered. This architecture is illustrated in Figure 5.13.

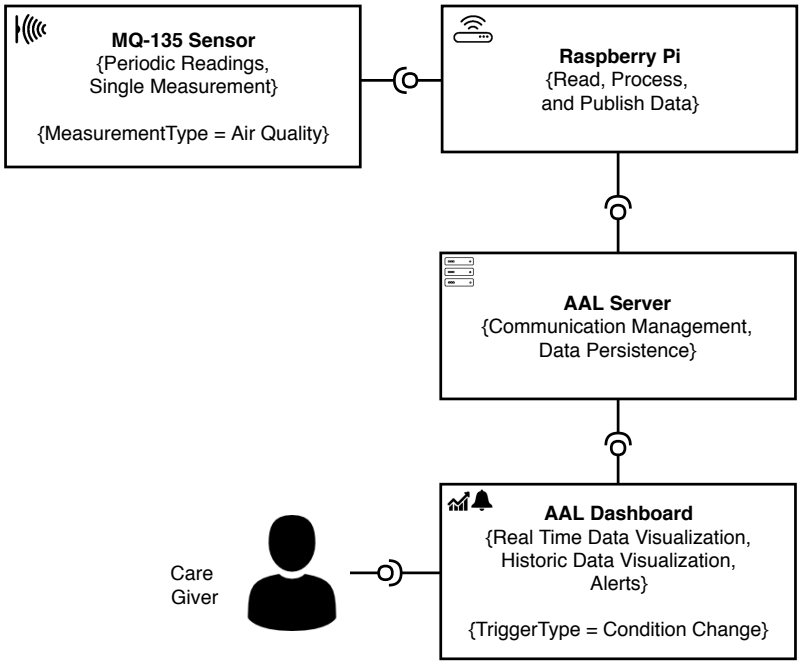


Figure 5.13: Architecture illustrating the test case for air quality monitoring features.

Test Patterns:

This test case involves testing the triggering of an alert following the reaching of a pre-set condition, and requires instantiating the following test patterns:

- Test Periodic Readings;
- Test Alerts.

5.2.4 Luminosity Test

Identifier:

TC_04

Name:

Luminosity Test

Description:

This test case considers that LDR sensor readings are collected periodically as an estimation of luminosity, which is used to trigger the activation and deactivation of the lights — via an actuator (the Sonoff Slampher). This behaviour is captured in Figure 5.14.

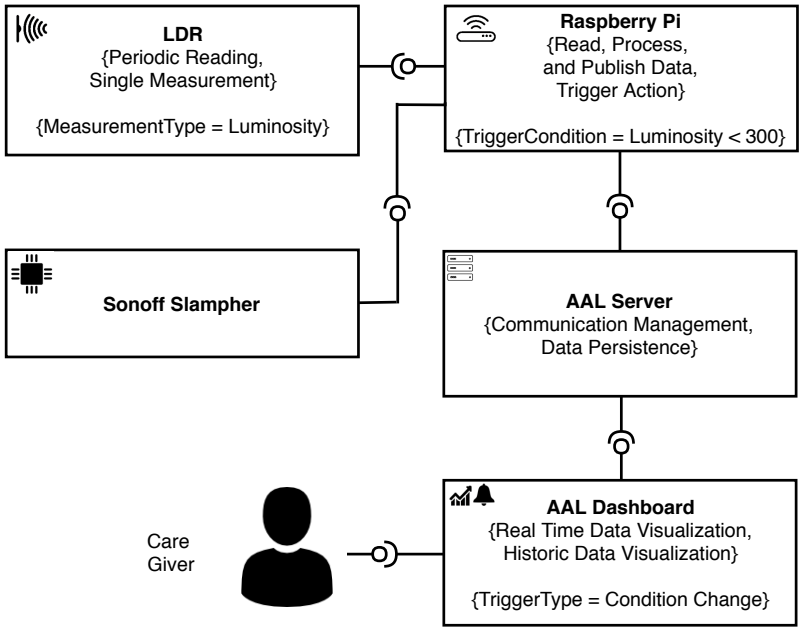


Figure 5.14: Architecture illustrating the test case for luminosity features.

Test Patterns:

This test case involves testing the triggering of an action following the reaching of a pre-set condition, and requires instantiating the following test patterns:

- Test Periodic Readings;
- Test Actuators;
- Test Actions.

5.2.5 Weight Monitoring Test

Identifier:

TC_05

Name:

Weight Monitoring Test

Description:

This test case considers that Nokia Body+ scale readings are triggered and collected as weight measurements, and that these measurements trigger an alert when their value is below or above pre-set levels. This functionality is captured in Figure 5.15.

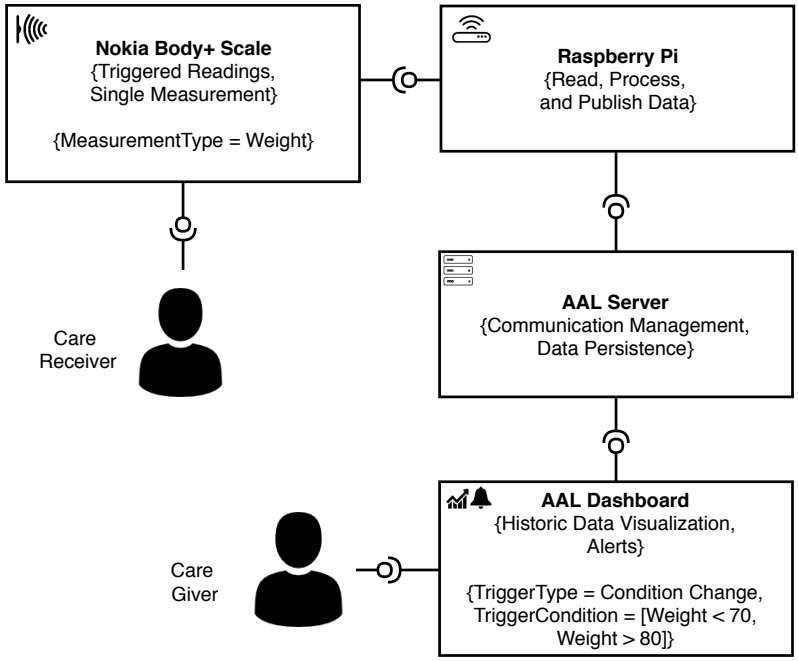


Figure 5.15: Architecture illustrating the test case for weight monitoring features.

Test Patterns:

This test case involves testing the triggering of readings and alerts following the reaching of a pre-set condition, and requires instantiating the following test patterns:

- Test Triggered Readings;
- Test Alerts.

5.2.6 Blood Pressure Monitoring Test

Identifier:

TC_06

Name:

Blood Pressure Monitoring Test

Description:

This test case considers that Nokia BPM readings are collected as heart rate and systolic/-diastolic pressure measurements, with an alert being generated when those readings indicate a problem. This functionality is captured in Figure 5.16.

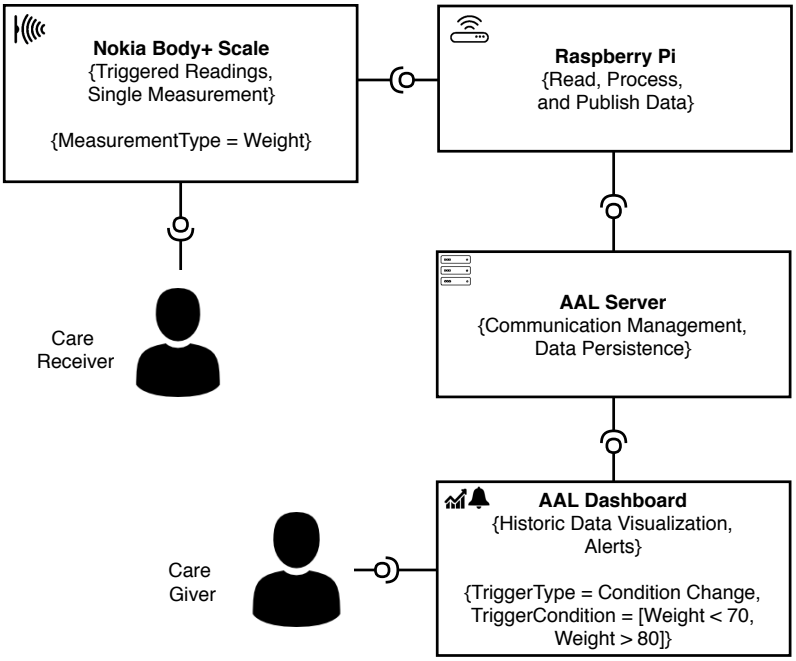


Figure 5.16: Architecture illustrating the test case for blood pressure monitoring features.

Test Patterns:

This test case involves testing the triggering of readings and alerts following the reaching of a pre-set condition, and requires instantiating the following test patterns:

- Test Triggered Readings;
- Test Alerts.

5.3 Results

Once set-up, the test cases described were put to test using the framework. Figure 5.17 illustrates the output of *Izinto* after being applied to the test case TC_01. Specifically, the test patterns for periodic readings, alerts, actuators, and actions were instantiated, and all the checks passed.

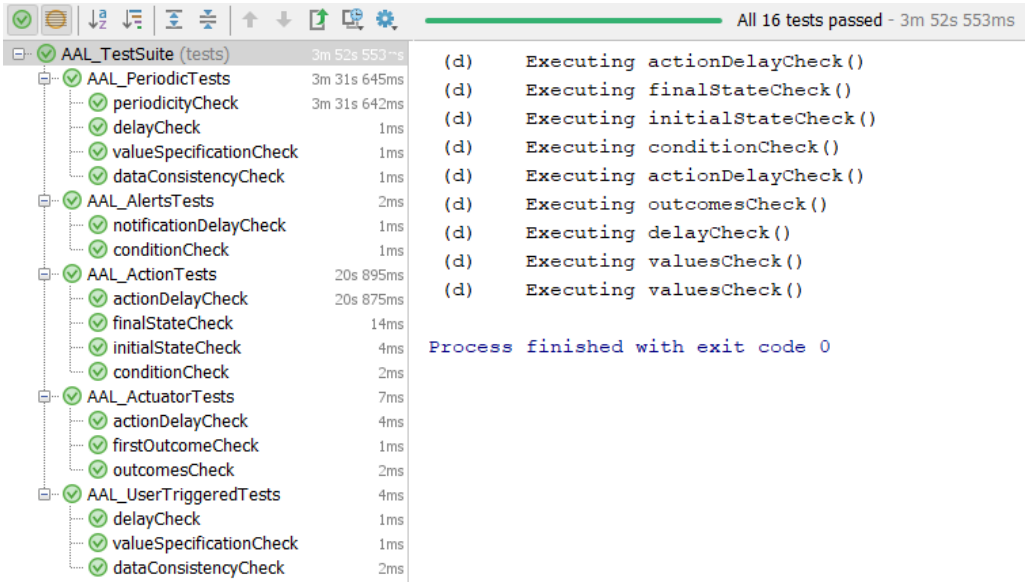


Figure 5.17: Example of output for a successful test run.

Further test cases were considered so as to validate the framework in various scenarios. The results of some of the tests performed are presented in Table 5.7, detailing the errors detected and their origin: while some errors actually resulted from real errors in the scenario's implementation, others were purposefully introduced, so as to ensure *Izinto*'s effectiveness.

Test Case	**Outcome**	**Error**	
		Type	**Description**
TC_01	Periodicity observed for readings of type Temperature exceeded the acceptable deviation	Injected Error	Periodicity set to below the sensor's response time
TC_03	Alert notification not received	Implementation Error	Incorrect email host configuration for alert module
TC_04	Action not executed as expected	Injected Error	Internet connectivity disrupted
TC_05	Readings of type Weight not compliant with value specification	Implementation Error	Mismatch in the measurement unit considered and that used by Nokia Health's API

Table 5.7: Outcome of different test cases considered.

The framework enabled the identification of errors by flagging, among others, issues with the readings performed — both with respect to their periodicity and compliance with value specification —, as well as alert notification delays, and issues with the triggering and/or execution of actions. Figure 5.18 reproduces the output of an unsuccessful test run, in which a periodicity check failed for test case TC_01, because the periodicity observed exceeded the acceptable deviation specified — the expected periodicity was set to an interval lower than the sensors response time.

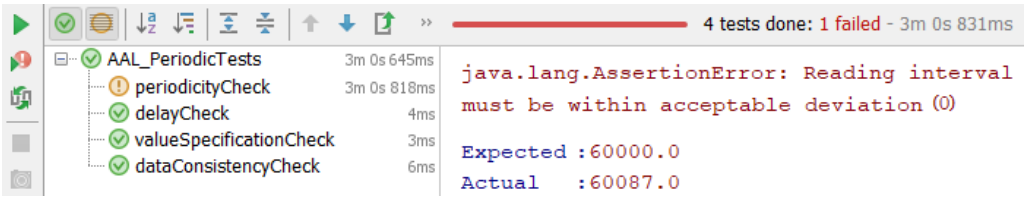


Figure 5.18: Example of output for an unsuccessful test run.

5.4 Discussion

The approach described and applied in testing this application scenario allowed to achieve automated integration testing of the IoT ecosystem. This pattern-based approach enhances reusability — as the test patterns can be applied to various scenarios to test recurring behaviours in the scope of IoT — and extendability — as it is possible to define other test patterns not identified as of yet.

The framework makes it possible to test recurring behaviours in the scope of IoT in an automated manner without the need for dealing with test logic. Furthermore, it has the potential to reduce the effort that needs to be put into the configuration of communication protocols for the various components that make up the SUT, by supporting the most widely used technologies out-of-the-box. The process of configuring the test of an IoT ecosystem may be subject to improvements in the future (see Section 6.3).

This validation consisted in simulating a scenario in which a system integrator builds an IoT application in a specific domain by integrating different components in order to implement a certain functionality and performs integration tests to ensure that the application delivers this functionality.

The framework enabled the identification of errors introduced both on purpose and by accident, proving its effectiveness. It flagged, among others, issues with the readings performed — both with respect to their periodicity and compliance with value specification —, as well as alert notification delays, and issues with the triggering and/or execution of actions.

5.5 Summary

This chapter described the validation of the framework proposed in Chapter 4. A concrete IoT application scenario in the domain of AAL was devised, implemented and explored, so as to cover various test cases. Furthermore, *Izinto*'s potential for reuse and extension — both with regard to the patterns implemented and the technologies supported —, along with its ability to detect errors were discussed.

Chapter 6

Conclusion

6.1 Overview **79**
6.2 Contributions **80**
6.3 Future Work **81**
6.3.1 IoT Test Patterns 81
6.3.2 IoT Testing Framework 81

This chapter presents the conclusions of this research, starting with an overview of the work covered in this dissertation, followed by a list of the contributions, and concluding with a set of possible routes for future work.

6.1 Overview

Current technological trends will likely drive the emergence of IoT solutions in the near future, across several domains. However, the difficulty to test IoT systems — due to the heterogeneous and distributed nature of the system — and the importance of testing in the development process give rise to the need for an efficient way to implement automated testing in IoT.

An overview of existing tools for IoT testing was carried out, and demonstrated that current solutions have not yet properly addressed the need for a test framework for automated integration testing of IoT ecosystems.

As part of the approach proposed, a feature model was devised, enabling the representation of the plurality of components and features of an IoT ecosystem. This was a first step in identifying a set of recurring test strategies for IoT applications, which can be considered test patterns specific to the IoT domain.

Further, a pattern-based test automation framework for integration testing of IoT ecosystems was developed. The framework implements — in a generic way and out-of-the-box — a set of test patterns specific to the IoT domain which can be easily instantiated for concrete IoT scenarios with minimal technical knowledge.

Finally, the framework proposed was validated within a concrete application scenario in the domain of AAL, which was put together and allowed the exploration of various test cases. This allowed to discuss the framework's ability to detect errors and ease of use, along with its potential for reuse and extension — both with regard to the patterns implemented and the technologies supported.

6.2 Contributions

The main contributions of this work include:

- A review of i) the state-of-the-art in IoT and the challenges involved in developing and testing IoT solutions, and ii) the existing tools for IoT testing;
- A feature model to categorize and group the components of an IoT ecosystem according to their characteristics and a set of test patterns that allow the test of recurring behaviours of IoT systems, which define a set of steps that should be taken into consideration when testing some specific behaviours of IoT systems;
- A pattern-based test automation framework, which aims to reduce IoT testing efforts. The framework supports the reuse and implementation of test patterns to automatically exercise recurring behaviours of IoT systems.

It is also worth noting that some of the work and ideas covered in this dissertation have been used in the writing of a scientific publication — entitled *A Pattern-based IoT Testing Framework* — which was submitted and accepted to the Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things [EI], part of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018) [201b]. Furthermore, a second article describing the test patterns identified — *Test Patterns for IoT* — was submitted for review to A-Test [Wor], a workshop co-located with the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering(ESEC/FSE 2018) [201a]. Both articles can be found in full in Appendix B.

In conclusion, it can be considered that the main goals of this work were attained.

6.3 Future Work

Future work can be pursued in two different directions: on a more theoretical level, it is possible to work toward the definition of additional IoT Test Patterns; from a more practical perspective, it is possible to further develop the framework.

6.3.1 IoT Test Patterns

The test patterns identified focus more on the test of features, corresponding to functional requirements. As such, it may be pertinent to identify different test patterns, that cover other aspects, such as connectivity, security, scalability, and performance, along with other non-functional requirements. This could be achieved, for instance, by applying Machine Learning techniques to recognize a set of recurring behaviors of IoT applications to which can correspond a specific test strategy. Another possibility is to create test patterns that match existing design patterns within the scope of IoT.

6.3.2 IoT Testing Framework

The development of a visual interface would facilitate the testing process, by simplifying the test configuration step — for instance, by introducing the possibility of generating the configurations for a certain SUT automatically — and improving the display of the test results. Finally, making the framework open-source would promote its extension so as to, on the one hand, implement new test patterns as they are identified, and, on the other hand, add support for other technologies.

Conclusion

References

[201a] ESEC/FSE 2018. Esec/fse 2018. https://2018.fseconference.org/home. Accessed: 2018-06-29.

[201b] ISSTA 2018. Issta 2018. https://conf.researchr.org/home/issta-2018. Accessed: 2018-06-22.

[ABF+15] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. FIT IoT-LAB: A large scale open experimental IoT testbed. In *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pages 459–464, 2015.

[ABF+16] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. Model-based testing as a service for IoT platforms. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9953 LNCS, pages 727–742, 2016.

[AFGM+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, 17(4):2347–2376, 2015.

[ALNT14] Mohammad A. Alsheikh, Shao W. Lin, Dusit Niyato, and Hp Tan. Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996–2018, 2014.

[ANR] ANRG. Cooja simulator - contiki. http://anrg.usc.edu/contiki/index.php/Cooja_Simulator. [Accessed on 28/01/2018].

[Are] Theo Arends. Sonoff tasmota. https://github.com/arendst/Sonoff-Tasmota. Accessed: 2018-04-29.

[Ash09] K Ashton. That 'Internet of Things' Thing. *RFiD Journal*, page 4986, 2009.

[Ass18a] American Heart Association. Understanding blood pressure readings. https://www.heart.org/HEARTORG/Conditions/HighBloodPressure/KnowYourNumbers/Understanding-Blood-Pressure-Readings_UCM_301764_Article.jsp, April, 2018. [Accessed on 5/4/2018].

[Ass18b] American Heart Association. Know your target heart rates for exercise, losing weight and health. https://healthyforgood.heart.org/move-more/articles/target-heart-rates, January, 2018. [Accessed on 20/4/2018].

[Bat05] Don Batory. Feature models, grammars, and propositional formulas. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3714 LNCS, pages 7–20, 2005.

[BHG+13] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas Schmidt. RIOT OS: Towards an OS for the Internet of Things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, 2013.

[Bis17] Subhasis Biswal. Internet of things (iot) testing: Challenges, tools and testing approach. http://www.softwaretestinghelp.com/internet-of-things-iot-testing/, December, 2017. [Accessed on 01/18/2018].

[BL14] Michael Blackstock and Rodger Lea. Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED). In *Proceedings of the 5th International Workshop on Web of Things - WoT '14*, pages 34–39, 2014.

[CCH15] F.-C. Chang, D.-K. Chen, and H.-C. Huang. Future classroom with the internet of things a service-oriented framework. *Journal of Information Hiding and Multimedia Signal Processing*, 6(5):869–881, 2015.

[CHA16] GULLENA SATISH CHANDRA. Pattern language for IoT applications. In *PATTERN LANGUAGES OF PROGRAMS CONFERENCE*, 2016.

[Col17] Louis Columbus. 2017 roundup of internet of things forecasts. https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts, December 2017. [Accessed on 01/07/2018].

[CPN14] Pedro Costa, Ana C R Paiva, and Miguel Nabuco. Pattern based GUI testing for mobile applications. In *Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014*, pages 66–74, 2014.

[DGV04] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - A lightweight and flexible operating system for tiny networked sensors. In *Proceedings - Conference on Local Computer Networks, LCN*, pages 455–462, 2004.

[Dim16] Dimiter V. Dimitrov. Medical internet of things and big data in healthcare, 2016.

[DK] Digi-Key. Am2302 wired dht22 temperature humidity sensor. https://www.digikey.com/catalog/en/partgroup/am2302-wired-dht22-temperature-humidity-sensor/59255. [Accessed on 19/03/2018].

[EI] ECOOP and ISSTA. Tav-cps/iot 2018 - ecoop and issta. https://conf.researchr.org/track/ecoop-issta-2018/tavcpsiot-2018-papers. Accessed: 2018-06-21.

[Ele] ElectroFun. Sensor de luz ldr gl5528. https://www.electrofun.pt/sensor-luz-ldr. [Accessed on 19/03/2018].

[ESA01] Department Of Economic and United Nations Social Affairs, Population Division. World population ageing: 1950-2050. http://www.un.org/esa/

population/publications/worldageing19502050/, 2001. [Accessed on 01/07/2018].

[FGM+99] Roy Fielding, James Gettys, Jeff Mogul, Henrik Frystyk, Larry Masinter, P. Leach, and Tim Berners-Lee. RFC2616 - Hypertext transfer protocol–HTTP/1.1. *Internet Engineering Task Force*, pages 1–114, 1999.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[Fita] Fitbit. Fitbit app & dashboard. https://www.fitbit.com/eu/app. [Accessed on 21/03/2018].

[Fitb] Fitbit. Fitbit charge 2 heart rate + fitness wristband. https://www.fitbit.com/charge2. [Accessed on 21/03/2018].

[Fitc] Fitbit. Fitbit web api. https://dev.fitbit.com/build/reference/web-api/. [Accessed on 21/02/2018].

[FOUa] RASPBERRY PI FOUNDATION. Raspberry pi 3 model b - raspberry pi. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Accessed: 2018-06-21.

[Foub] The Eclipse Foundation. iot.eclipse.org - iot development made simple. https://iot.eclipse.org. [Accessed on 01/20/2018].

[Fun] Software Testing Fundamentals. Integration testing - software testing fundamentals. http://softwaretestingfundamentals.com/integration-testing/. [Accessed on 07/01/2018].

[Ger17] Hristo Gergov. Testing iot — taking the first steps in testing internet of things devices. https://huddle.eurostarsoftwaretesting.com/testing-iot-taking-the-first-steps-in-testing-internet-of-things-devices/, June, 2017. [Accessed on 01/18/2018].

[Ger18] Anna Gerber. Connecting all the things in the internet of things. https://www.ibm.com/developerworks/library/iot-lp101-connectivity-network-protocols/index.html, January, 2018. [Accessed on 01/16/2018].

[GKN+11] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. A survey on facilities for experimental internet of things research, 2011.

[GRC+14] L. A. Grieco, A. Rizzo, S. Colucci, S. Sicari, G. Piro, D. Di Paola, and G. Boggia. IoT-aided robotics applications: Technological implications, target domains and open issues. *Computer Communications*, 54:32–47, 2014.

[Guc] Sam Guckenheimer. What is continuous integration? https://www.visualstudio.com/learn/what-is-continuous-integration/. [Accessed on 01/18/2018].

[Gup] Harshit Gupta. Github - harshitgupta1337/fogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. https://github.com/harshitgupta1337/fogsim. [Accessed on 01/02/2018].

[GVGB17] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. In *Software - Practice and Experience*, volume 47, pages 1275–1296, 2017.

[Hl08] Comparing Hl. The HL7 Evolution. *corepoint HEALTH*, 1:318, 2008.

[HS02] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

[IKK+15] S M Riazul Islam, Daehan Kwak, Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The Internet of Things for Health Care : A Comprehensive Survey. *Access, IEEE*, 3:678 – 708, 2015.

[IL] FIT IoT-LAB. Fit/iot-lab – very large scale open wireless sensor network testbed. https://www.iot-lab.info/. [Accessed on 31/01/2018].

[Ind] Adafruit Industries. Dht sensor library. https://github.com/adafruit/DHT-sensor-library. Accessed: 2018-03-09.

[iS] i SCOOP. Internet of things (iot) in healthcare: benefits, use cases and evolutions. https://www.i-scoop.eu/internet-of-things-guide/internet-things-healthcare/. [Accessed on 01/15/2018].

[ITE16a] ITEAD. Sonoff s20 smart socket: Wifi controlled smart plug. https://www.itead.cc/smart-socket.html, 2016. Accessed: 2018-05-14.

[ITE16b] ITEAD. Sonoff slampher: Wifi smart led light bulb socket holder | itead. https://www.itead.cc/slampher.html, 2016. Accessed: 2018-05-14.

[ITU05] ITU. The Internet of Things. *Itu Internet Report 2005*, page 212, 2005.

[JUn] JUnit. Junit - about. https://junit.org/junit4/. Accessed: 2018-05-24.

[Kaa] Kaa. Using kaa endpoint sdks - kaa. https://kaaproject.github.io/kaa/docs/v0.10.0/Programming-guide/Using-Kaa-endpoint-SDKs/. Accessed: 2018-03-21.

[KBG13] Dmitry G. Korzun, Sergey I. Balandin, and Andrei V. Gurtov. Deployment of smart spaces in internet of things: Overview of the design challenges. In Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networking*, pages 48–59, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[LF16] B Lima and J P Faria. Towards the Online Testing of Distributed and Heterogeneous Systems with Extended Petri Nets. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 230–235, 2016.

[LL03] Philip Levis and Nelson Lee. Tossim: a simulator for tinyos networks. 12 2003.

[LML+10] Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, 2010.

[LODYJ13] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Yla-Jaaski. MAMMOTH: A massive-scale emulation platform for Internet of Things. In *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, volume 3, pages 1235–1239, 2013.

[MCW+16] Jorge Miranda, Jorge Cabral, Stefan Rahr Wagner, Christian Fischer Pedersen, Blaise Ravelo, Mukhtiar Memon, and Morten Mathiesen. An open platform for seamless sensor support in healthcare for the internet of things. *Sensors (Switzerland)*, 16(12), 2016.

[MD97] Gerard Meszaros and Jim Doble. Pattern languages of program design 3. chapter A Pattern Language for Pattern Writing, pages 529–574. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Mer06] Vassili Van Der Mersch. Automated testing for the internet of things. https://nordicapis.com/automated-testing-for-the-internet-of-things/, May, 2006. [Accessed on 01/18/2018].

[MHAK17] N Mekki, M Hamdi, T Aguili, and T.-H. Kim. Scenario-based vulnerability analysis in IoT-based patient monitoring system. *ICETE 2017 - Proceedings of the 14th International Joint Conference on e-Business and Telecommunications*, 4(January):554–559, 2017.

[MHF17] Brice Morin, Nicolas Harrand, and Franck Fleurey. Model-Based Software Engineering to Tame the IoT Jungle. *IEEE Software*, 34(1):30–36, 2017.

[MK17] Nitinder Mohan and Jussi Kangasharju. Edge-Fog cloud: A distributed cloud for Internet of Things computations. In *2016 Cloudification of the Internet of Things, CIoT 2016*, 2017.

[MP15] Ines Coimbra Morgado and Ana C R Paiva. Test Patterns for Android Mobile Applications. *Proceedings of the 20th European Conference on Pattern Languages of Programs*, pages 1–7, 2015.

[MPM13] Rodrigo M L M Moreira, Ana C R Paiva, and Atif Memon. A pattern-based approach for GUI modeling and testing. In *2013 IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013*, pages 288–297, 2013.

[Mur] Matthew Murdoch. Github - mmurdoch/arduinounit: Arduinounit is a unit testing framework for arduino libraries. https://github.com/mmurdoch/arduinounit. [Accessed on 01/02/2018].

[Mur13] Matthew Murdoch. Arduinounit. https://github.com/mmurdoch/arduinounit, 2013. Accessed: 2018-05-10.

[Noka] Nokia. Nokia body+ | wi-fi body analyzer scale. https://health.nokia.com/pt/en/body-plus. [Accessed on 21/03/2018].

[Nokb] Nokia. Nokia health api developer documentation. https://developer.health.nokia.com/api/. [Accessed on 21/02/2018].

[Nokc] Nokia. Nokia health mate | total health tracking: Activity, heart, weight, sleep. https://health.nokia.com/pt/en/health-mate. [Accessed on 21/03/2018].

[Nor16] Amy Nordrum. Popular internet of things forecast of 50 billion devices by 2020 is outdated. https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated, 2016.

[OAS14] OASIS. MQTT Version 3.1.1. *OASIS Standard*, 2014.

[ÖDE+06] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with COOJA. In *Proceedings - Conference on Local Computer Networks, LCN*, pages 641–648, 2006.

[Ora] Oracle. Java 8 central. http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html. Accessed: 2018-06-25.

[Org] Personal Connected Health Alliance A HIMSS Organization. Continua design guidelines. http://www.pchalliance.org/continua-design-guidelines. [Accessed on 01/24/2018].

[Pat14] Paolo Patierno. Iot protocols landscape. https://www.slideshare.net/paolopat/io-t-protocols-landscape, June, 2014. [Accessed on 01/16/2018].

[Pla] PlatformIO. An open source ecosystem for iot development – platformio. http://platformio.org/. [Accessed on 02/02/2018].

[Pra17] Prabhu. Iot automation – how feasible it is? http://www.indiumsoft.com/Blog/iot-automated-testing/, August, 2017. [Accessed on 01/20/2018].

[PSCW10] Anneke Pehmöller, Frank Salger Capgemini, and Stefan Wagner. Patterns for testing in global software development. 10 2010.

[PTRa] PTRobotics. Espressif esp32. https://www.ptrobotics.com/wifi/6256-espressif-esp32-devkitc.html. Accessed: 2018-03-22.

[PTRb] PTRobotics. Ftdi basic breakout - 3.3v. https://www.ptrobotics.com/conversores/1445-ftdi-basic-breakout-33v.html. Accessed: 2018-04-29.

[PTRc] PTRobotics. Mq-135 gas sensor. https://www.ptrobotics.com/gases/4144-mq-135-gas-sensor.html. Accessed: 2018-04-13.

[PZCG14] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):414–454, 2014.

[RBF+16] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. Internet of things patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs - EuroPlop '16*, pages 1–21, 2016.

[RBF+17] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. Internet of Things Patterns for Devices. In *Proceedings of the 9th International Conferences on Pervasive Patterns and Applications - PATTERNS 2017*, pages 117–126, 2017.

[RDD+17] António Ramadas, Gil Domingues, João Pedro Dias, Ademar Aguiar, and Hugo Sereno Ferreira. Patterns for Things that Fail. In *24th Conference on Pattern Languages of Programs (PLoP 2017)*, 2017.

[Red] Redash. Redash helps you make sense of your data | redash. https://redash.io/. Accessed: 2018-04-07.

[RTG+15] Amir Mohammad Rahmani, Nanda Kumar Thanigaivelan, Tuan Nguyen Gia, Jose Granados, Behailu Negash, Pasi Liljeberg, and Hannu Tenhunen. Smart e-Health Gateway: Bringing intelligence to Internet-of-Things based ubiquitous healthcare systems. In *2015 12th Annual IEEE Consumer Communications and Networking Conference, CCNC 2015*, pages 826–834, 2015.

[RWBO15] Philipp Rosenkranz, Matthias Wählisch, Emmanuel Baccelli, and Ludwig Ortmann. A Distributed Test System Architecture for Open-source IoT Software. In *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems - IoT-Sys '15*, pages 43–48, 2015.

[SDFC16] Sergio Saponara, Massimiliano Donati, Luca Fanucci, and Alessio Celli. An Embedded Sensing and Communication Platform, and a Healthcare Model for Remote Monitoring of Chronic Diseases. *Electronics*, 5(3):47, 2016.

[SHB13] Z Shelby, K. Hartke, and C. Bormann. Constrained Application Protocol(CoAP). *CoRE Working Group*, pages 1–118, 2013.

[Sim] Inc SimpleSoft. Simplesoft's iot simulator for coap, mqtt, mqtt-sn, http/rest sensors and gateways. http://www.smplsft.com/SimpleIoTSimulator.html. [Accessed on 28/01/2018].

[SK16] Thirumalasetty Sivakanth and S. Kolangiammal. Design of Iot based smart health monitoring and alert system. *International Journal of Control Theory and Applications*, 9(15):7655–7661, 2016.

[SK17] Sidra Siddiqui and Tamim Ahmed Khan. On Test Patterns for Cloud Applications. In *Proceedings - 14th International Conference on Frontiers of Information Technology, FIT 2016*, pages 57–62, 2017.

[SKH+15] John Soldatos, Nikos Kefalakis, Manfred Hauswirth, Martin Serrano, Jean-Paul Calbimonte, Mehdi Riahi, Karl Aberer, Prem Prakash Jayaraman, Arkady Zaslavsky, Ivana Podnar Žarko, Lea Skorin-Kapov, and Reinhard Herzog. Openiot: Open source internet-of-things in the cloud. In Ivana Podnar Žarko, Krešimir Pripužić, and Martin Serrano, editors, *Interoperability and Open-Source Solutions for the Internet of Things*, pages 13–25, Cham, 2015. Springer International Publishing.

[S.L] IoTIFY Technologies S.L. Iotify- develop full stack iot application with virtual device simulation. https://iotify.io/. [Accessed on 01/02/2018].

[SLMN15] Robert Stackowiak, Art Licht, Venu Mantha, and Louis Nagode. *Big Data and The Internet of Things: Enterprise Information Architecture for A New Age*. Apress, Berkely, CA, USA, 1st edition, 2015.

[SM14] H Sulaiman and A I Magaireah. Factors affecting the adoption of integrated cloud-based e- health record in healthcare organizations: a case study of Jordan. In *Information Technology and Multimedia (ICIMU), 2014 International Conference on*, pages 102–107, 2014.

[Smi] Albert Smith. Top iot tools and platforms for developers - hidden brains blog. http://www.hiddenbrains.com/blog/top-iot-tools-and-platforms-for-developers.html. [Accessed on 22/01/2018].

[Spo17] Alexander Spotnitz. Moving the cloud to the edge. https://www.pubnub.com/blog/moving-the-cloud-to-the-edge-computing/, June, 2017. [Accessed on 01/16/2018].

[SS17] Pallavi Sethi and Smruti R. Sarangi. Internet of Things: Architectures, Protocols, and Applications, 2017.

[Sta01] Statista. Iot devices installed base worldwide 2015-2025 | statistic. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide, 2001. [Accessed on 01/07/2018].

[Sta10] International Standard. *ISO/IEC/IEEE Health informatics–Personal health device communication–Part 20601: Application profile–Optimized exchange protocol*, volume December, 2010. IEEE, 2010.

[Tan10] Lu Tan. Future internet: The Internet of Things. *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, pages V5–376–V5–380, 2010.

[Teca] TechMartian. Interfacing photoresistor with esp32. http://www.instructables.com/id/Interfacing-Photoresistor-With-ESP32/. [Accessed on 19/03/2018].

[Tecb] KaaIoT Technologies. Kaa open-source iot platform — iot cloud platform the internet of things solutions and applications that set the standard. https://www.kaaproject.org. [Accessed on 01/23/2018].

[TLCY14] Chun Wei Tsai, Chin Feng Lai, Ming Chao Chiang, and Laurence T. Yang. Data mining for internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):77–97, 2014.

[TM17] Antero Taivalsaari and Tommi Mikkonen. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software*, 34(1):72–80, 2017.

[VFG+09] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, Pat Doody, Friess Peter, Guillemin Patrick, Gusmeroli Sergio, Bassi Harald, Sundmaeker Alessandro, Jubert Ignacio Soler, Mazura Margaretha, Harrison Mark, Eisenhauer Markus, and Doody Pat. Internet of Things Strategic Research Roadmap. *Internet of Things Strategic Research Roadmap*, pages 9–52, 2009.

[VPZ12] Sokratis Vavilis, Milan Petkovi, and Nicola Zannone. Impact of ICT on Home Healthcare ICT innovation in Healthcare. In *Eindhoven University of Technology*, page 12, 2012.

[Wäh16] Kai Wähner. Project flogo: Golang-powered open source iot integration framework. https://dzone.com/articles/project-flogo-golang-powered-open-source-iot-integ, November, 2016. [Accessed on 01/22/2018].

[Wor] The A-Test 2018 Workshop. The a-test 2018 workshop. https://a-test.org/. Accessed: 2018-06-29.

[Wor01] World Health Organization. Innovative care for chronic health conditions. Technical Report 1, World Health Organization, 2001.

[YD11] Mustafa Yuksel and Asuman Dogac. Interoperability of medical device information and the clinical applications: An HL7 RMIM based on the ISO/IEEE 11073 DIM. *IEEE Transactions on Information Technology in Biomedicine*, 15(4):557–566, 2011.

[YXM+14] Geng Yang, Li Xie, Matti Mäntysalo, Xiaolin Zhou, Zhibo Pang, Li Da Xu, Sharon Kao-Walter, Qiang Chen, and Li Rong Zheng. A Health-IoT platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box. *IEEE Transactions on Industrial Informatics*, 10(4):2180–2191, 2014.

REFERENCES

Appendix A

Configuration File Example

Excerpt A.1 is an example of a configuration file, with the settings for testing the scenario described in Section 5.1 for validation purposes.

```
{
 "testDuration": 300000,
 "dataSources": [{
        "type": "MQTTBroker",
        "specs": {
            "mqttBrokerAddress": "tcp://10.227.107.152",
            "topics": ["aal/#"]
        }
    },
    {
        "type": "REST_API",
        "specs": {
            "url": "10.227.107.152/readings",
            "headers": {
                "Authorization" : ##################################
            }
        }
    }],
 "devices": [{
        "type": "Sensor",
        "specs": {
            "id": "am-2302",
            "readingSettings": [{
                "type": "Temperature",
                "expectedInterval": 60000,
                "acceptableDeviation": 50,
                "acceptableDelay": 2000,
                "valueSpecification": {"minimumValue": -40,
                                       "maximumValue": 80,
                                       "readingTrend": {"type":"Decreasing"},
                                       "readingInductor": {
```

```
                                        "id": "sonoff-s20",
                                        "commands": [{
                                            "message": "Deactivating fan.",
                                            "interval": 60000,
                                            "command": "cmnd/sonoff-s20/power",
                                            "args": "OFF"
                                        },
                                        {
                                            "message": "Activating fan.",
                                            "interval": 60000,
                                            "command": "cmnd/sonoff-s20/power",
                                            "args": "ON"
                                        },
                                        {
                                            "message": "Deactivating fan.",
                                            "command": "cmnd/sonoff-s20/power",
                                            "args": "OFF"
                                        }]
                                    }
                                }
            },
            {
                 "type": "Humidity",
                 "expectedInterval": 120000,
                 "acceptableDeviation": 50,
                 "acceptableDelay": 2000,
                 "valueSpecification": {"minimumValue": 0,
                                        "maximumValue": 100}
             }]
        }
    },
    {
        "type": "Sensor",
        "specs": {
            "id": "mq135_01",
            "readingSettings": [{
                "type": "Air Quality",
                "expectedInterval": 60000,
                "acceptableDeviation": 6000,
                "acceptableDelay": 2000,
                "valueSpecification": {"minimumValue": 0,
                                       "maximumValue": 500000}
            }]
        }
    },
    {
        "type": "Sensor",
        "specs": {
            "id": "LDR",
```

```
            "readingSettings": [{
                "type": "Luminosity",
                "expectedInterval": 60000,
                "acceptableDeviation": 6000,
                "acceptableDelay": 2000,
                "valueSpecification": {"minimumValue": 0,
                                       "maximumValue": 500}
            }]
        }
    },
    {
       "type": "Sensor",
        "specs": {
            "id": "nokia_body+scale",
            "readingSettings": [{
                "type": "Weight",
                "acceptableDelay": 120000,
                "valueSpecification": {"minimumValue": 0,
                                       "maximumValue": 200,
                                       "readingInductor": {
                                            "type": "Human",
                                            "specs": {
                                                "commands": [{
                                                    "message": "Trigger weight
                                                        reading.",
                                                }]
                                            }
                                        }
                                    }
            }]
        }
    },
    {
        "type": "Sensor",
        "specs": {
            "id": "nokia_bpm",
            "readingSettings": [{
                "type": "Diastolic Pressure",
                "acceptableDelay": 60000,
                "valueSpecification": {"minimumValue": 50,
                                       "maximumValue": 200,
                                       "readingInductor": {
                                            "type": "Human",
                                            "specs": {
                                                "commands": [{
                                                    "message": "Trigger blood
                                                        pressure reading."
                                                }]
                                            }
```

```
                                        }
                                    }
                },
                {
                    "type": "Systolic Pressure",
                    "acceptableDelay": 60000,
                    "valueSpecification": {"minimumValue": 50,
                                           "maximumValue": 200}
                }]
        }
    },
    {
        "type": "MQTTActuator",
        "specs": {
            "id": "sonoff-s20",
            "mqttBrokerAddress": "tcp://10.227.107.152",
            "stateTopic": "stat/sonoff-s20/POWER",
            "resetCommand": {
                "message": "Turn off socket",
                "command": "cmnd/sonoff-s20/power",
                "args": "OFF"
            }
        }
    },
    {
        "type": "MQTTActuator",
        "specs": {
            "id": "sonoff-slampher",
            "mqttBrokerAddress": "tcp://10.227.107.152",
            "stateTopic": "stat/sonoff-slampher/POWER",
            "resetCommand": {
                "message": "Turn off lightbulb holder",
                "command": "cmnd/sonoff-slampher/power",
                "args": "OFF"
            }
        }
    }],
"triggers": [{
        "type": "TriggerAll",
        "specs": {
            "conditions": [{
                "type": "Higher",
                "specs": {
                    "sensorID": "am-2302",
                    "type": "Temperature",
                    "value": 25
                }
        }],
        "outcomes": [{
```

```
            "type": "Alert",
            "specs": {
                "description": "Temperature > 25",
                "acceptableDelay": 10000,
                "subscribers": [{
                    "type": "EmailSubscriber",
                    "specs": {
                        "email": "kaahealth@gmail.com"
                    }
                }]
            }
        },
        {
            "type": "Action",
            "specs": {
                "description": "Turn ON AC",
                "acceptableDelay": 2000,
                "expectedInitialState": "OFF",
                "expectedFinalState": "ON",
                "actionChecker": {
                    "id": "sonoff-s20"
                }
            }
        }]
    }
},
{
    "type": "TriggerAll",
    "specs": {
        "conditions": [{
            "type": "Lower",
            "specs": {
                "sensorID": "am-2302",
                "type": "Temperature",
                "value": 25
            }
        }],
        "outcomes": [{
            "type": "Action",
            "specs": {
                "description": "Turn OFF AC",
                "acceptableDelay": 2000,
                "expectedInitialState": "ON",
                "expectedFinalState": "OFF",
                "actionChecker": {
                    "id": "sonoff-s20"
                }
            }
        }]
```

```
    }
},
{
    "type": "TriggerAll",
    "specs": {
        "conditions": [{
            "type": "Lower",
            "specs": {
                "sensorID": "ldr",
                "type": "Luminosity",
                "value": 300
            }
        }],
        "outcomes": [{
            "type": "Action",
            "specs": {
                "description": "Turn ON lightbulb",
                "acceptableDelay": 2000,
                "expectedInitialState": "OFF",
                "expectedFinalState": "ON",
                "actionChecker": {
                    "id": "sonoff-slampher"
                }
           }
        }]
    }
},
{
  "type": "TriggerAll",
  "specs": {
      "conditions": [{
          "type": "Higher",
          "specs": {
              "sensorID": "ldr",
              "type": "Luminosity",
              "value": 300
          }
      }],
      "outcomes": [{
          "type": "Action",
          "specs": {
              "description": "Turn OFF lightbulb",
              "acceptableDelay": 2000,
              "expectedInitialState": "ON",
              "expectedFinalState": "OFF",
              "actionChecker": {
                  "id": "sonoff-slampher"
              }
         }
```

```
        }]
    }
},
{
    "type": "TriggerAll",
    "specs": {
        "conditions": [{
            "type": "Higher",
            "specs": {
                "sensorID": "mq-135",
                "type": "Air Quality",
                "value": 1000
            }
        }],
        "outcomes": [{
            "type": "Alert",
            "specs": {
                "description": "Poor air quality",
                "acceptableDelay": 10000,
                "subscribers": [{
                    "type": "EmailSubscriber",
                    "specs": {
                        "email": "kaahealth@gmail.com"
                    }
                }]
            }
        }]
    }
},
{
    "type": "TriggerAny",
    "specs": {
        "conditions": [{
            "type": "Higher",
            "specs": {
                "sensorID": "nokia-body+scale",
                "type": "Weight",
                "value": 80
            }
        },
        {
            "type": "Lower",
            "specs": {
                "sensorID": "nokia-body+scale",
                "type": "Weight",
                "value": 70
            }
        }],
        "outcomes": [{
```

```
            "type": "Alert",
            "specs": {
                "description": "Weight requires attention",
                "acceptableDelay": 10000,
                "subscribers": [{
                    "type": "EmailSubscriber",
                    "specs": {
                        "email": "kaahealth@gmail.com"
                    }
                }]
            }
        }]
    }
},
{
    "type": "TriggerAll",
    "specs": {
        "conditions": [{
            "type": "Lower",
            "specs": {
                "sensorID": "nokia-bpm",
                "type": "Diastolic Pressure",
                "value": 80
            }
        },
        {
            "type": "Higher",
            "specs": {
                "sensorID": "nokia-bpm",
                "type": "Systolic Pressure",
                "value": 120
            }
        }],
        "outcomes": [{
            "type": "Alert",
            "specs": {
                "description": "Elevated Blood Pressure",
                "acceptableDelay": 10000,
                "subscribers": [{
                    "type": "EmailSubscriber",
                    "specs": {
                        "email": "kaahealth@gmail.com"
                    }
                }]
            }
        }]
    }
},
{
```

```
    "type": "TriggerAll",
    "specs": {
        "conditions": [{
            "type": "Higher",
            "specs": {
                "type": "Heart Rate",
                "value": 120
            }
        }],
        "outcomes": [{
            "type": "Alert",
            "specs": {
                "description": "Elevated Heart Rate",
                "acceptableDelay": 10000,
                "subscribers": [{
                    "type": "EmailSubscriber",
                    "specs": {
                        "email": "kaahealth@gmail.com"
                    }
                }]
            }
        }]
    }
 }]
}
```

Excerpt A.1: Settings for testing the scenario considered.

Appendix B

Scientific Publications

B.1 *Izinto*: A Pattern-based IoT Testing Framework

The work and ideas covered in Chapter 4 have been used in the writing of a scientific publication, submitted and accepted to the Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things [EI], part of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018) [201b].

Izinto: A Pattern-Based IoT Testing Framework

Pedro Martins Pontes, Bruno Lima, João Pascoal Faria
Faculty of Engineering, University of Porto
INESC TEC
Rua Dr. Roberto Frias, s/n 4200-465, Porto, Portugal
{pedro.martins.pontes,bruno.lima,jpf}@fe.up.pt

ABSTRACT

The emergence of Internet of Things (IoT) technology is expected to offer new promising solutions in various domains and, consequently, impact many aspects of everyday life. However, the development and testing of software applications and services for IoT systems encompasses several challenges that existing solutions have not yet properly addressed. Particularly, the difficulty to test IoT systems — due to their heterogeneous and distributed nature —, and the importance of testing in the development process give rise to the need for an efficient way to implement automated testing in IoT. Although there are already several tools that can be used in the testing of IoT systems, a number of issues can be pointed out, such as focusing on a specific platform, language, or standard, limiting the possibility of improvement or extension, and not providing out-of-the-box functionalities. This paper describes *Izinto*, a pattern-based test automation framework for integration testing of IoT systems. The framework implements in a generic way a set of test patterns specific to the IoT domain which can be easily instantiated for concrete IoT scenarios. It was validated in a number of test cases, within a concrete application scenario in the domain of Ambient Assisted Living (AAL).

KEYWORDS

Internet of Things, Test Patterns, Testing Framework

1 INTRODUCTION

Internet of Things (IoT) envisions a reality of pervasive connectivity, in which all *things* will be connected to the Internet and interact with the physical environment through sensors and actuators, collecting and exchanging data to feed services and intelligence, resulting in a fusion between the physical and digital worlds [6]. This technology is expected to impact our professional, personal and social environments, by allowing the development of new promising solutions in a variety of domains, including, but not limited to, agriculture, healthcare, utilities, and transportation, pushing towards the realization of other concepts, such as smart cities and smart homes [13, 24].

Although different forecasts predict different growth rates, they all show a rapid growth over the coming years, both in terms of market value and the number of connected devices. Despite the fact that some predictions can be viewed as unrealistic, it is safe to estimate that the number of IoT connected devices worldwide will soon surpass the 10 billion mark [20]. Taking into consideration the range and scale of IoT applications, and given that IoT applications will become an integral part of everyday life, some failures can have dire consequences, making it paramount to ensure their correctness.

However, there are challenges associated with the development and testing of IoT applications and services. These include problems such as the lack of reference architectures and protocols, heterogeneity and the lack of standardization — leading to interoperability issues —, faulty development strategies and insufficient automation, and security and privacy issues [15]. Particularly, the number and diversity of individual components and the distributed nature of IoT systems pose major challenges to their testing, debugging, and validation.

Currently, there are a number of solutions for testing IoT systems, which follow different testing approaches, focus different test levels (Unit, Integration, System, and Acceptance Testing [5]), and cover different layers [18] (see Figure 1).

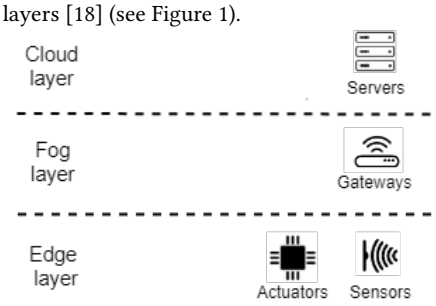


Figure 1: Typical layered composition of an IoT system.

Nonetheless, it is possible to identify several limitations in existing solutions [16], which include failing to account for the heterogeneity of the IoT field — by focusing on a specific platform, language or standard —, lacking the possibility of improvement or functionality extension — for instance, by not making source code publicly available — and not providing out-of-the-box functionalities [7].

In this article, we describe a pattern-based test automation framework for integration testing of IoT ecosystems which was named *Izinto*. The framework implements — in a generic way and out-of-the-box — a set of test patterns specific to the IoT domain which can be easily instantiated for concrete IoT scenarios with minimal technical knowledge. Further, we demonstrate its validation in a number of test cases, within a concrete application scenario in the domain of Ambient Assisted Living (AAL).

The rest of the paper is organized as follows: Section 2 introduces a feature model for an IoT ecosystem and the set of test patterns implemented by the framework are identified and briefly described; Section 3 provides an overview of the framework developed and its architecture; in Section 4, a description of an example application scenario is presented, along with various test cases considered; related work is presented in Section 5; finally, conclusions and future work are laid out in Section 6.

2 TEST PATTERNS FOR IOT

2.1 Feature Model

By its own nature, the field of IoT involves a wide variety of systems and devices with very diverse features and, consequently, any testing approach must take into consideration these differences.

The components of an IoT ecosystem can, generally speaking, be categorized and grouped according to their characteristics. For this, we propose a model that allows representing the plurality of components and their features within an IoT ecosystem, based on the formalism of feature models [4], partially depicted in Figure 2.

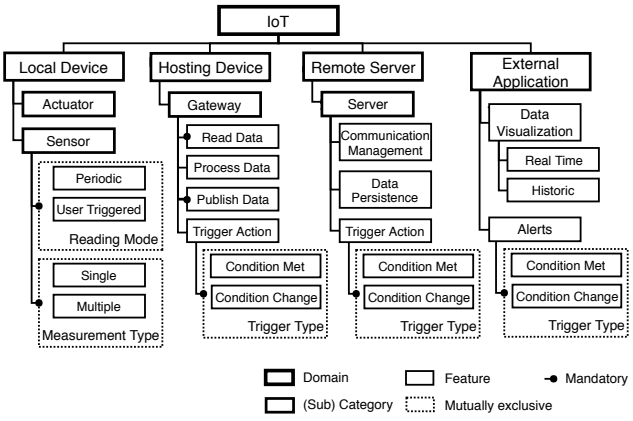


Figure 2: Partial feature model of an IoT ecosystem.

This model considers four main categories of components: local devices, hosting devices, remote servers, and external applications. Within some of these main categories, subcategories were defined.

A local device can be either a sensor or an actuator. A sensor can be categorized in accordance to its reading mode, i.e., whether readings are performed automatically, in a periodic fashion and without human interaction, or, conversely, if readings are triggered by a user. Additionally, it is possible to differentiate sensors based on the ability to measure various parameters as opposed to a single parameter.

The category of hosting devices corresponds to gateways, which are defined by features such as reading, processing, and publishing data, and potentially trigger an action (either each time a condition is met, or each time a condition status changes).

A remote server may support communication management and data persistence and, optionally, the triggering of actions.

External applications can support data visualization (either real time data or an historic record), and the generation of alerts (either each time a condition is met, or each time a condition status changes).

By identifying the different components of an IoT system and classifying each component in terms of its features, it is possible to determine which functionalities must be tested and to select which test(s) pattern(s) should be applied accordingly.

Pattern	Description
Test Periodic Readings	Check whether a sensor is capable of performing readings at a fixed rate and that those readings are correctly transmitted and persisted within the IoT system
Test Triggered Readings	Check whether a sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system
Test Actuators	Check whether an actuator is capable of executing commands, changing its state accordingly
Test Alerts	Check whether alerts are sent to pre-set subscribers whenever a pre-set condition is met
Test Actions	Check whether pre-set actions are executed whenever a pre-set condition is met

Table 1: Test patterns implemented.

2.2 Test Patterns

Taking into consideration the specificities of IoT systems, it is possible to identify a set of test strategies to test recurring behaviours of IoT systems which can be described as test patterns, as outlined in Table 1.

Lets consider the test of periodic readings as an example. This pattern addresses the need to ensure that a sensor is capable of performing readings at a fixed rate and that those readings are correctly transmitted and persisted within the IoT system, under normal operation. Table 2 presents a summarized definition for this test pattern, which presents the set of necessary checks in a more structured manner, along with a list of steps that would need to be taken into consideration to perform those checks.

First, the sensor might require some configuration — for instance, specifying a shorter reading interval to expedite the test.

Once set-up has been performed, the test itself would start, running for an appropriate amount of time, which should allow the collection of — at the very least — two readings. Upon the end of the test, it would be necessary to verify if valid readings were persisted within the IoT system, at the expected rate, accounting for possible transmission delays, by checking the timestamps of reading and reception.

In some cases, it may be desirable to also cross-check the readings persisted with some log kept on a gateway. Furthermore, it may be of interest to induce some variation in the parameter being measured. As such, in some cases, it may be possible to use an actuator to provoke variations in some parameter, or, in the cases in which that is not possible, have a person provoking these variations, with the test tool providing instructions at the appropriate time.

These procedures correspond to a recurring behaviour that can be applied whenever the need to test of periodic readings arises. A similar rationale was used in defining the other patterns.

Pattern	Test Periodic Readings
Objectives	- Check that readings are performed with the pre-set periodicity, with acceptable deviations - Check that the readings are transmitted with a delay within a pre-set maximum - Check if the values are correctly transmitted - Check that variations in the parameter being measured are observed
Procedures	- Set-up the sensor, so as to set an adequate rate for the readings to be performed - Induce some variation in the parameter being measured, by using an actuator or human input - Verify if valid readings are persisted within the IoT system, at the expected rate, accounting for possible transmission delays, by checking the timestamps of reading and reception.

Table 2: Pattern for testing periodic readings, defined by its objectives and procedures.

3 TEST FRAMEWORK

Implementing test cases covering multiple aspects, interfaces and protocols so as to ensure functional correctness is a demanding task, made worse by the heterogeneous and distributed nature of IoT systems.

To address this, we developed *Izinto* — a framework that makes it possible to test IoT solutions in an automated manner and with reduced effort, by implementing a set of test patterns out-of-the-box.

Requiring only a configuration file specifying the appropriate settings for the system under test (SUT), those with minimal technical knowledge about testing can still use *Izinto*. These specifications will depend on the tests which are to be performed and must include the settings for the devices which make up the SUT, as well as the settings for the data sources considered. To test actions and alerts, information on the triggers and expected outcomes must also be provided.

The framework also accounts for the possibility of configuring a *reading inductor*, so as to either manipulate the readings performed by the actual sensor (for instance, by using an actuator) or to inject readings directly into the system. Excerpts of such file are provided in Section 4 for a specific scenario.

In terms of its architecture — depicted in Figure 3 —, it is possible to distinguish two main modules: one module which encompasses the actual test logic (package TestHandlers) and another module corresponding to abstractions of IoT components (other packages).

The test logic was implemented using JUnit [12] and follows the set of patterns listed in Section 2.2. The second module is formed by a set of connectors for the actual IoT components — sensors, actuators and data sources —, which allow to configure and control said components.

Devices are identified by an ID, and are abstracted through adapters for the actual sensors and actuators, allowing their configuration. Sensors are characterized by a set of reading settings — one per each different type of measurement a particular sensor

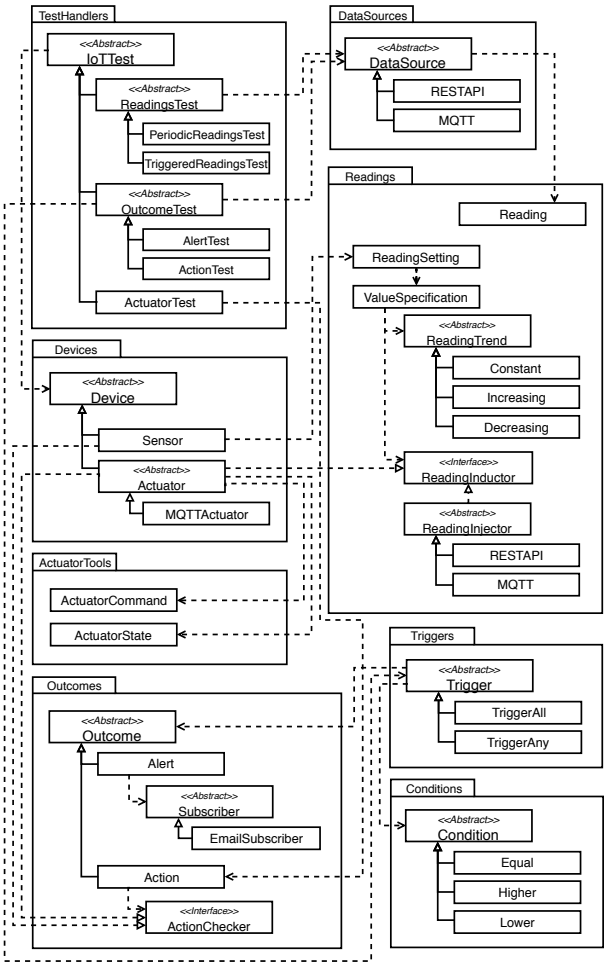


Figure 3: Simplified representation of the framework's architecture.

supports — which make it possible to detect a number of common data quality problems [9].

In the case of actuators, these adapters also allow to control the devices, through commands.

As of now, *Izinto* implements the patterns listed in Section 2.2 and supports the most widely used technologies with regard to devices and data sources — specifically, devices supporting MQTT [21] communications, and MQTT and REST API [8] as data sources. Having been developed as a modular framework, *Izinto* can be extended, as other test patterns emerge or as test experts encounter the need for supporting other technologies.

4 APPLICATION SCENARIO

The *Izinto* framework was validated within an IoT scenario in the domain of ambient assisted living (AAL), illustrated in Figure 4. The scenario considers a care receiver — the subject being monitored in a certain controlled environment —, and a care giver — the person monitoring the patient. Various sensors collect and transmit data

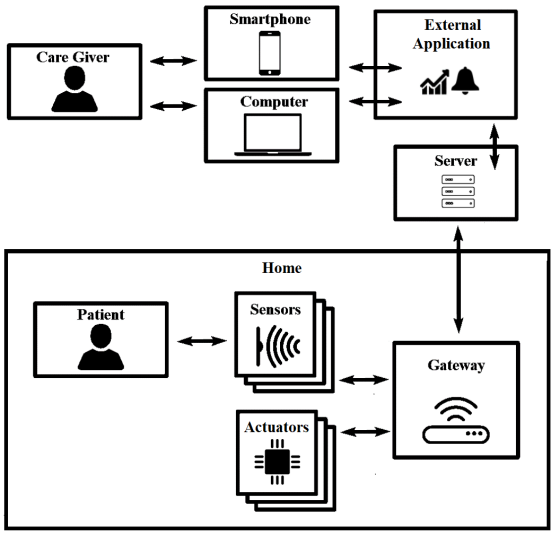


Figure 4: Diagram depicting the AAL scenario used for validating the framework.

both on the patient's health parameters and on ambient conditions to a server. This server maintains a personal health record and triggers pre-determined actions and alerts according to the condition of the patient or the surrounding environment, notifying a care giver so that it can be determined if additional measures must be taken.

In the scenario considered, there are body sensors — a blood pressure monitor, a weighing scale, and a fitness bracelet (which includes a heartbeat sensor) —, and ambient sensors — sensors for temperature, humidity, air quality, and luminosity. Actuators were also included, specifically a lightbulb holder and a smart socket connected to an air conditioner (AC) unit. A set of *triggers* were configured, so as to perform actions — such as turning on a fan when the temperature rises above a certain temperature, or turning on a light when the luminosity drops below a certain threshold —, as well as to generate alerts — for instance, when the humidity drops below a certain threshold or a reading indicates a dangerous blood pressure level.

It is possible to instantiate different test patterns in different test architectures. One such architecture is illustrated in Figure 5, which includes different components, inserted into the respective categories and characterized by their features, allowing the instantiation of several test patterns, namely the patterns to test periodic readings, test alerts, and test actions. This particular case pertains to testing the triggering of an alert and action following the reaching of a pre-set condition. In this example, the AM2302 sensor is capable of transmitting (passively) periodic readings of multiple types — Temperature and Humidity —; a gateway collects, processes and transmits those readings to a server — which handles communications, data management, and the triggering of an action (through an actuator), based on the condition $Temperature > 28$, the same condition used by an external application to generate alerts.

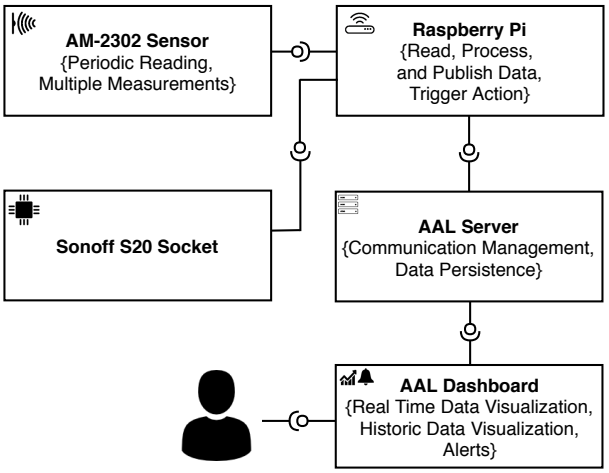


Figure 5: Example of test architecture for covering various categories and features.

```
{"type": "Sensor",
 "specs": {"id": "AM2302", "readingSettings":
          [{"type": "Temperature",
            "expectedInterval": 60000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": -40,
                                   "maximumValue": 80}},
           {"type": "Humidity",
            "expectedInterval": 120000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": 0,
                                   "maximumValue": 100}}]
}}
```

Excerpt 1: Configuration for testing periodic temperature and humidity readings (test periodic readings pattern).

```
{"type": "TriggerAll",
 "specs": {"conditions": [{"type": "Higher",
           "specs": {"sensorID": "AM2302",
                     "type": "Temperature",
                     "value": 28
           }}],
           "outcomes": [{"type": "Alert",
           "specs": {"description": "Temperature > 28",
                     "acceptableDelay": 10000,
                     "subscriber": {
                           "type": "EmailSubscriber",
                           "specs": {
                           "email": "kaahealth@gmail.com"
                     }}
           }}]
}}
```

Excerpt 2: Configuration for testing a temperature alert (test alerts pattern).

In this case, excerpt 1 corresponds to part of the configuration file required for testing a sensor, expected to perform temperature readings every minute and humidity readings every two minutes, with a maximum deviation of 3 seconds and a maximum transmission delay of 2 seconds. The sensor's specification states it is capable of measuring temperatures from -40ºC up to 80ºC and that it measures the relative humidity from 0% up to 100%.

In turn, excerpt 2 corresponds to the part of the configuration file required for testing the triggering of an alert, which must be sent once via email when the temperatures rises above 28ºC.

Once set-up, the scenario was put to test using *Izinto*. Figure 6 illustrates the output from running a set of periodicity tests within the scenario described. In this instance, all tests passed, with the exception of the periodicity check, since the periodicity observed exceeded the acceptable deviation.

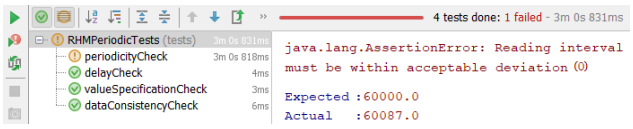


Figure 6: Example of output of periodicity tests.

Further tests were performed, so as to validate the framework in various scenarios. Table 3 describes a few of these tests and presents the corresponding results. The *Izinto* framework enabled the identification of errors by flagging, among others, issues with the readings performed — both with respect to their periodicity and compliance with value specification —, as well as alert notification delays, and issues with the triggering and/or execution of actions.

Test Case	Test Pattern	Outcome
MQ-135 sensor readings are periodically collected as Air Quality measurements	Test Periodic Readings	Periodicity observed exceeded the acceptable deviation
Blood pressure monitor readings are collected as Heart Rate and Systolic/Diastolic pressure measurements	Test Triggered Readings	Readings of type Systolic/Diastolic pressure not compliant with value specification
Control the lights by switching a lightbulb holder on and off	Test Actuators	No failures detected
Heart Rate measurements above a pre-set level trigger an alert	Test Alerts	Alert notification delayed beyond acceptable
Temperature measurements above a pre-set level trigger the activation of the AC	Test Actions	Action not executed

Table 3: Examples of different test cases considered.

5 RELATED WORK

There are already some solutions available for the purpose of testing IoT systems focusing different IoT layers and enabling technologies, both in terms of commercial solutions and academic work.

Some software development and testing tools — such as ArduinoUnit [19] and PlatformIO [22] — can be used for testing IoT applications, often relying on physical devices to conduct the testing, but focus mainly on the level of Unit Testing.

Following an approach that resorts to a combination of model-based testing (MBT) techniques and service oriented solutions, Ahmad et. al conceived MBTAAS [2] which allows to systematically test IoT and data platforms.

Another possible approach to test IoT-based applications consists in using systems that behave like the SUT — emulators and/or simulators. TOSSIM [14] is a wireless sensor network simulator, built with the specific goal of simulating TinyOS devices, supporting two programming interfaces — Python and C++ —, and allowing various levels of simulation, from hardware interrupts to high-level system events, such as packet arrivals. Similarly, COOJA [3] is an extensible Java-based simulation/emulation platform developed for the Contiki operating system. It provides a complete simulator for sensor node software, where the network, operating system and even the machine code instruction set are simulated, allowing developers to test their code and systems before running it on the target hardware. COOJA has support for radio link simulation, with recently added support for a more complex radio interference simulation from Wi-Fi and Bluetooth. Looga et al. presented an emulation platform for IoT — MAMMotH [17]. Its architecture presumes three distinct scenarios based on GPRS communications. This is part of an ongoing project, with the ultimate goal of supporting nodes in the scale of tens of millions, and there are questions that are still open and need to be investigated in the future. iFogSim [11] is a simulator able to simulate edge devices, cloud data centers, and network links, and perform metrics evaluation on them, allowing the investigation and comparison of resource management techniques based on Quality of Service (QoS).

While some tests can be conducted on device emulators or simulators, other tests might need to be performed on real devices, in order to ensure that the combination of software and hardware works as specified [23]. Hence, it may be necessary to perform these tests in a network consisting of real devices, in a set-up similar to that of the SUT. To address this need and to make it possible to test IoT systems as a whole, work has been pursued towards the development of IoT testbeds. A number of physical testbeds are already active and publicly available [10], such as FIT IoT-LAB [1], a testbed that provides a large scale platform to test applications across the different layers. Although these tend to cover multiple IoT layers, they usually focus on a specific domain of application, and creating and maintaining these systems can be both complex and expensive.

Despite the fact that there are several solutions for testing IoT systems, following different testing approaches and focusing different test levels, it is possible to identify several limitations in existing solutions. For a number of solutions presented in the literature — as it is the case with MBTAAS —, it is not possible to find actual tools or frameworks to support implementation, making it complex to

adopt and use them. Other tools — including TOSSIM, and COOJA — focus on a specific platform, language, or standard, failing to account for the heterogeneity of the IoT field. Furthermore, most do not allow for the possibility of improvement or functionality extension — for instance, by not making source code publicly available —, and do not provide out-of-the-box functionalities.

The framework developed aims at reducing the effort put into testing IoT applications, by providing a set of out-of-the-box automated tests based on a various IoT test patterns, while allowing for the future extension of the functionalities offered.

6 CONCLUSIONS AND FUTURE WORK

The IoT paradigm has become a trend over the recent years. If, on the one hand, the heterogeneous and distributed nature of IoT systems makes it difficult to test IoT-based solutions, on the other hand, the inherent ubiquity of such applications evidences the importance of testing, giving rise to the need for an efficient and effective way to implement automated testing in IoT.

Although some solutions are already available for the purpose of testing IoT systems, it is possible to identify several limitations, such as those related to the lack of support for different IoT-enabling technologies, the lack of out-of-the-box functionalities and the impossibility of improvement by means of extension.

In this article we presented *Izinto* — a pattern-based IoT testing framework which reduces the effort put into testing IoT applications. Because it implements a set of test patterns, the framework developed provides out-of-the box functionalities that correspond to recurring behaviours within the scope of IoT testing.

Izinto was validated within a concrete scenario, corresponding to a AAL scheme in which various sensors collect and transmit data both on a patient's health parameters and on ambient conditions. Alerts are generated when the patient's condition is determined to require attention and pre-set actions are triggered by changes in ambient conditions.

Izinto is aimed at two different profiles. On the one hand, it can be used by testing experts to put virtually any IoT solution to the test, as the framework can be extended so as to meet a particular need. On the other hand, those with minimal technical knowledge about testing can still use the framework, as only some knowledge about the SUT is required to configure it.

As future work, we intend to develop a visual interface which facilitates the testing process, improving the test configuration process — for instance, by introducing the possibility of generating the configurations for a certain SUT automatically — and improving the display of the test results. Additionally, it may be pertinent to implement other test patterns, yet to be identified. Finally, we intend to make *Izinto* open-source to promote its extension so as to add support for other technologies.

REFERENCES

[1] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*. 459–464. https://doi.org/10.1109/WF-IoT.2015.7389098

[2] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. 2016. Model-based testing as a service for IoT platforms. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9953 LNCS. 727–742. https://doi.org/10.1007/978-3-319-47169-3_55

[3] Ba Bagula and Zenville Erasmus. 2015. Iot emulation with Cooja. In *Workshop on Scientific Applications for the Internet of Things ICTP*.

[4] Don Batory. 2005. Feature models, grammars, and propositional formulas. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 3714 LNCS. 7–20. https://doi.org/10.1007/11554844_3

[5] Boris Beizer. 2003. *Software Testing Techniques*. 550 pages. http://books.google.com/books?id=Ixf97h356zcC

[6] L Coetzee and J Eksteen. 2011. The Internet of Things - promise for the future? An introduction. *IST-Africa Conference Proceedings, 2011* (2011), 1–9. https://doi.org/ISBN:978-1-905824-24-3

[7] João Pedro Dias, Flávio Couto, Ana C.R. Paiva, and Hugo Sereno Ferreira. 2018. A Brief Overview of Existing Tools for Testing the Internet-of-Things.

[8] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation. AAI9980887.

[9] Ralf Gitzel, Subanatarajan Subbiah, and Christopher Ganz. 2018. A Data Quality Dashboard for CMMS Data. *ICORES 2018* Icores (2018), 170–177. https://doi.org/10.5220/0006552501700177

[10] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. 2011. A survey on facilities for experimental internet of things research. , 58–67 pages. https://doi.org/10.1109/MCOM.2011.6069710 arXiv:1609.07712

[11] Harshit Gupta, Amir VahidDastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. In *Software - Practice and Experience*, Vol. 47. 1275–1296. https://doi.org/10.1002/spe.2509 arXiv:1606.02007

[12] JUnit. [n. d.]. JUnit - About. https://junit.org/junit4/. Accessed: 2018-05-24.

[13] Dmitry G. Korzun, Sergey I. Balandin, and Andrei V. Gurtov. 2013. Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges. In *Internet of Things, Smart Spaces, and Next Generation Networking*, Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 48–59.

[14] Philip Levis and Nelson Lee. 2003. TOSSIM: a simulator for TinyOS networks. (12 2003).

[15] Shancang Li, Li Da Xu, and Shanshan Zhao. 2015. The internet of things: a survey. *Information Systems Frontiers* 17, 2 (2015), 243–259. https://doi.org/10.1007/s10796-014-9492-7 arXiv:arXiv:1011.1669v3

[16] Bruno Lima and João Pascoal Faria. 2017. A Survey on Testing Distributed and Heterogeneous Systems: The State of the Practice. In *Software Technologies*, Enrique Cabello, Jorge Cardoso, André Ludwig, Leszek A. Maciaszek, and Marten van Sinderen (Eds.). Springer International Publishing, Cham, 88–107.

[17] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Yla-Jaaski. 2013. MAMMOTH: A massive-scale emulation platform for Internet of Things. In *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, Vol. 3. 1235–1239. https://doi.org/10.1109/CCIS.2012.6664581

[18] Nitinder Mohan and Jussi Kangasharju. 2017. Edge-Fog cloud: A distributed cloud for Internet of Things computations. In *2016 Cloudification of the Internet of Things, CIoT 2016*. https://doi.org/10.1109/CIOT.2016.7872914 arXiv:1702.06335

[19] Matthew Murdoch. 2013. ArduinoUnit. https://github.com/mmurdoch/arduinounit. Accessed: 2018-05-10.

[20] Amy Nordrum. 2016. Popular internet of things forecast of 50 billion devices by 2020 is outdated. https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[21] OASIS. 2014. MQTT Version 3.1.1. *OASIS Standard* October (2014), 81. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[22] PlatformIO. 2018. An open source ecosystem for IoT development – PlatformIO. https://platformio.org/. Accessed: 2018-05-16.

[23] Philipp Rosenkranz, Matthias Wählisch, Emmanuel Baccelli, and Ludwig Ortmann. 2015. A Distributed Test System Architecture for Open-source IoT Software. In *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems - IoT-Sys '15*. 43–48. https://doi.org/10.1145/2753476.2753481

[24] Lu Tan. 2010. Future internet: The Internet of Things. *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)* (2010), V5–376–V5–380. https://doi.org/10.1109/ICACTE.2010.5579543

B.2 Test Patterns for IoT

The patterns described in Chapter 3 are the basis of a scientific publication, submitted to the A-TEST workshop [Wor], co-located with the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering(ESEC/FSE 2018) [201a].

Test Patterns for IoT

Pedro Martins Pontes, Bruno Lima, João Pascoal Faria
Faculty of Engineering, University of Porto
INESC TEC
Rua Dr. Roberto Frias, s/n 4200-465, Porto, Portugal
{pedro.martins.pontes,bruno.lima,jpf}@fe.up.pt

ABSTRACT

The Internet of Things (IoT) is expected to bring forward new promising solutions in various domains. Consequently, it can impact many aspects of everyday life, and errors can have dramatic consequences. However, the absence of standard testing processes and methods poses a major challenge for IoT testing, driving developers and test experts to test IoT solutions using particular strategies and following different methodologies to address similar needs. Nonetheless, a closer examination allows to identify a set of recurring behaviors of IoT applications and a set of corresponding test strategies. This paper formalizes the notion of a Pattern-Based IoT Testing method for systematizing and automating the testing of IoT ecosystems by defining a set of test strategies for recurring behaviours of the IoT system, which can be defined as IoT Test Patterns: Test Periodic Readings, Test Triggered Readings, Test Alerts, Test Actions, and Test Actuators.

KEYWORDS

Internet of Things, IoT Patterns, Pattern-Based Testing, Test Patterns

1 INTRODUCTION

The Internet of Things (IoT) can be defined as a paradigm in which *things* can act as sensors and/or actuators and communicate to achieve a specific purpose [14], and is expected to bring forward new promising solutions in various domains. It has been estimated that the number of IoT connected devices worldwide will soon surpass the 10 billion mark [7] and, consequently, IoT will impact many aspects of everyday life. Because IoT poses as one of the largest and most widespread systems, errors can have a direct and dramatic impact in everyday life.

Nonetheless, the heterogeneous and distributed nature of IoT systems makes the process of testing IoT applications a challenging activity. Although there are already a number of solutions for testing IoT applications, these tend to focus on a specific platform, language, or standard, hinder improvement or extension, and not provide out-of-the-box functionality.

The lack of comprehensive solutions to test IoT systems can lead to the adoption of poor testing practices. This absence of standard testing processes and methods poses a major challenge for IoT testing, driving developers and test experts to test IoT solutions create individual strategies and follow different methodologies to address similar needs. Nonetheless, a closer examination allows to identify a set of recurring behaviors of IoT applications and a set of corresponding test strategies.

In this article, we propose a pattern-based approach to IoT testing and identify five IoT test patterns corresponding to a set of recurring behaviors of IoT applications and a set of corresponding test strategies. By defining these behaviours and strategies in a more formal manner it becomes possible to facilitate the process of testing IoT application, as these test patterns can be applied in different scenarios to test recurring behaviours in the scope of IoT.

The rest of the article is organized as follows: Section 2 provides an overview of related work, focusing on pattern-based approaches to design and testing; Section 3 presents the IoT test patterns identified; in Section 4 we discuss the usefulness and applicability of the IoT Test Patterns described; finally, the conclusions are drawn and future work is laid out in Section 5.

2 BACKGROUND AND RELATED WORK

The technological aspects of IoT systems and the details of their implementation reveal recurring solutions for the common problems of the field, which can be described as design patterns. Although there is still a large number of recurrent solutions in the scope of IoT yet to be documented, efforts have already been made towards the definition and categorization of design patterns for IoT-based systems [1]. Initial contributions in the context of defining these design patterns established patterns representing some key aspects of the IoT domain were made by Reinfurt et al. [12, 13]. Further work has been pursued in defining patterns that address how to deal with highly-changeable, error-prone and failing devices, and their networks [11].

Pattern-based test approaches have been followed in various domains [9]. These approaches are based on the assumption that systems similar in design, i.e., that follow the same set of design patterns, should share a similar test strategy.

Siddiqui and Khan [15] proposed a pattern-based technique for testing cloud applications and identified a set of test patterns by studying what features this type of application would support. For the purpose of evaluation, they considered threats to cloud applications and discussed the applicability of these test patterns.

Moreira et. al [5] put forward a pattern-based approach for Graphical User Interface (GUI) modeling and testing in order to systematize and automate the testing process. The notion of User Interface (UI) test pattern is introduced, corresponding to the association of a set of test strategies to a UI pattern. Each UI test pattern may be instantiated so as to represent the minutely different implementations and check if a test passed or failed.

Recognizing the need to develop test strategies specific for the mobile world, Costa et. al [2] performed an experiment to assess if the same approach could be used to test mobile applications, which produced encouraging results. Morgado and Paiva [6] extend this set of UI test patterns by identifying additional UI test patterns specific to the testing of mobile applications.

We were unable to find in the literature any reference to pattern-based IoT Testing and, to the best of our knowledge, there is no

work concerning this topic. Because there is no work focusing on the definition of test patterns specific to the IoT ecosystems, the goal of this research work is to realize a pattern-based approach by defining some IoT test patterns specific for the testing of IoT applications.

3 THE IOT TEST PATTERNS

Having analyzed a number of IoT solutions in different domains — including in ambient assisted living (AAL) [3] and home automation [8]. Taking into consideration the particularity of IoT systems, it is possible to recognize a set of test strategies to test recurring behaviours of IoT systems which can be described as test patterns. As such, we introduce the notion of IoT Test Pattern, which is the association of a set of test strategies to an IoT pattern. Five different test patterns were identified, as listed in Table 1.

Pattern	Description
Test Periodic Readings	Check whether a sensor is capable of performing readings at a fixed rate and that those readings are correctly transmitted and persisted within the IoT system
Test Triggered Readings	Check whether a sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system
Test Actuators	Check whether an actuator is capable of executing commands, changing its state accordingly
Test Alerts	Check whether alerts are sent to pre-set subscribers whenever a pre-set condition is met
Test Actions	Check whether pre-set actions are executed whenever a pre-set condition is met

Table 1: IoT test patterns identified.

For instance, the test pattern for Periodic Readings defines a strategy to test IoT system in which a sensor is expected to perform periodic readings. However, the implementation of this functionality may differ in the different IoT ecosystems, and, as such, the pattern to Test Periodic Readings may be configure so as to describe slightly different implementations and verify if the various checks defined passed or failed.

In order to make these patterns reusable and to facilitate the definition of more patterns, a template based on the one proposed by Meszaros and Doble [4] was used. Each pattern is distinguished by a name and further described through various fields, defined as follows:

- ***Name***: unique identifier to shortly refer the pattern;
- ***Context***: situation where the problem occurs;
- ***Problem***: description of the problem addressed by the pattern;
- ***Solution***: description of the proposed solution for the pattern;
- ***Example***: example where the (IoT Test) pattern can be applied.

Next, we described the IoT test patterns identified following this template.

3.1 Test Periodic Readings

Name:

Test Periodic Readings

Context:

Under the paradigm of IoT, all *things* will be connected to the Internet and interact with the physical environment through sensors, collecting and exchanging data. Being equipped with sensing capability, these devices can periodically perform measurements on a number of parameters and transmit this data — either actively or passively, depending on their degree of autonomy.

Problem:

It must be ensured that a sensor is capable of performing readings at a fixed rate and that those readings are correctly transmitted and persisted within the IoT system, under normal operation.

Therefore, it is necessary to:

- Check that the readings are performed with the pre-set periodicity, with acceptable deviations;
- Check that the readings are transmitted with a delay below a pre-set maximum;
- Check that the readings are correctly transmitted;
- Check that the readings are valid and, optionally, that variations are observed.

Solution:

The test pattern for this type of behavior is only applied when a sensor supports the periodic execution of readings and consists of the following steps:

(1) Setup the sensor(s);
This may include specifying the type of readings to be performed and setting an adequate rate for each type.
(2) Setup a gateway (optional);
This may include configuring it to maintain a log of the data collected, so that it can be compared with the data persisted.
(3) Induce some variation in the parameter(s) being measured (optional);
In some cases, it may be possible to use an actuator to produce those variations. In the cases in which that is not feasible, it may be possible to resort to human input to produce those variations.
(4) Verify if valid readings are persisted within the IoT system, at the expected rate, accounting for possible transmission delays.
The validity of the readings relates to their value — specifically, whether pre-set thresholds and/or trends are observed — and to the consistency across the several points of the IoT system considered. Checking if the readings are performed

with the pre-set periodicity and accounting for possible deviations and transmission delays may be done by checking the timestamps of reading and reception, assuming that the different components' clocks are synchronized.

It should be noted that the total runtime for the test must be defined taking into account the periodicity of the readings, so as to ensure that a sufficient number of measurements are performed.

Example:

Take an IoT system which consists of temperature and air quality sensors, which are setup to periodically perform measurements of the environment the system is in. The temperature sensors are expected to perform temperature readings every five minutes and are capable of measuring temperatures from -40ºC up to 80ºC. The air quality sensors measure the level of Carbon Monoxide (CO) and Carbon Dioxide (CO_2) every minute, in Parts Per Million (PPM). Using this pattern it becomes possible to ensure that all sensors perform these readings at the expected rate and that those readings are correctly transmitted and persisted within the IoT system. This pattern can be instantiated simply by providing a set of inputs, including the test runtime, sensor settings — with regard to id, type of readings performed, expected periodicity, acceptable deviation and delay, and expected value ranges —, and the means for accessing the data collected.

3.2 Test Triggered Readings

Name:

Test Triggered Readings

Context:

IoT devices are meant to work in concert for and with people at home, in industry or in the enterprise. In a number of situations, users are expected to trigger readings, commanding the sensor to perform measurements on a number of parameters and to transmit this data — either actively or passively, depending on their degree of autonomy.

Problem:

It must be ensured that a sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system, under normal operation.

Therefore, it is necessary to:

- Check that the readings are transmitted with a delay below a pre-set maximum;
- Check that the readings are correctly transmitted;
- Check that the readings are valid and, optionally, that variations are observed.

Solution:

The test pattern for this type of behavior is only applied when a sensor supports the triggering of readings and consists of the following steps:

(1) Setup the sensor(s);
This may include specifying the type of readings to be performed.
(2) Setup a gateway (optional);
This may include configuring it to maintain a log of the data collected, so that it can be compared with the data persisted.
(3) Trigger the readings;
In some cases, it might be possible to use an actuator to trigger the reading, or — in the cases in which that is not feasible —, have a human do it.
(4) Induce some variation in the parameter(s) being measured (optional);
In some cases, it may be possible to use an actuator to produce those variations. In the cases in which that is not feasible, it may be possible to resort to human input to produce those variations.
(5) Verify if valid readings are persisted within the IoT system, at the expected rate, accounting for possible transmission delays.
The validity of the readings relates to their value — specifically, whether pre-set thresholds and/or trends are observed — and to the consistency across the several points of the IoT system considered. Checking if the readings are performed and transmitted within the maximum acceptable delay may be done by checking the timestamps of reading and reception, assuming that the different components' clocks are synchronized.

It should be noted that the total runtime for the test must be defined taking into account the time necessary to trigger the readings, and in a way that allows a sufficient number of measurements to be performed.

Example:

Take an IoT system which includes a blood pressure monitor, which can be used to check diastolic and systolic pressure, as well as heart rate. Using this pattern it becomes possible to ensure that this sensor is capable of performing measurements triggered by a user and that those readings are correctly transmitted and persisted within the IoT system, by providing a set of inputs, including the test runtime, sensor settings — with regard to id, type of readings performed, acceptable delay, and expected value ranges —, and the means for accessing the data collected.

3.3 Test Alerts

Name:

Test Alerts

Context:

Many IoT applications involve generating alerts whenever a certain condition is met. These alerts and conditions are usually dependent on the data collected by the sensors.

Problem:

It must be ensured that an alert is triggered and that all subscribers are notified, under normal operation.

Therefore, it is necessary to:

- Check the conditions for triggering the alert are met;

- Check that a notification reaches all the subscribers, within an acceptable delay.

Solution:

The test pattern for this type of behavior is only applied when an alert is expected to be generated when a specific condition — or set of conditions — dependent on readings collected by one or more sensors is met. It consists of the following steps:

(1) Generate readings capable of triggering the expected alert(s);
    This can be attained either by using physical sensors and possibly actuators to produce readings within certain thresholds. This would involve:
    (a) Setup the sensor(s);
        This may include specifying the type of readings to be performed.
    (b) Setup a gateway (optional);
    Alternatively, this could be achieved without using actual sensors, but by injecting data directly to the IoT system.
(2) Verify if the readings persisted within the IoT system are capable of triggering the expected alert;
(3) Verify if all the subscribers to the alert(s) are notified.
    It is important to account for possible delays between the triggering of the alert(s) and the reception of the notification(s). The maximum acceptable delay will depend on the nature of the application and the technology used to produce the actual notifications (email, SMS, etc.).

Example:

Take again an IoT system which includes a blood pressure monitor, which can be used to diastolic and systolic pressure, as well as heart rate. Consider now that the system is configured to issue an alert when the blood pressure is either too high or too low, notifying a doctor who can then provide advice on how to proceed. This pattern makes it possible to ensure that the alert is issued when the blood pressure readings indicate a problem, and that the doctor is notified, by providing a set of inputs, which includes the test runtime, sensor settings — with regard to id and type of readings performed —, the means for accessing the data collected, and the details about the alert itself — with respect to the triggering condition(s), the notification and the subscribers.

3.4 Test Actions

Name:

Test Actions

Context:

Several IoT applications involve triggering the execution of some action whenever a certain condition is met. As with the case of alerts, these conditions are usually dependent on the data collected by the sensors.

Problem:

It must be ensured that a pre-set action is triggered and executed, under normal operation.

Therefore, it is necessary to:

- Check that the expected action has been triggered;
- Check that the action was performed with a maximum delay, ensuring that the actuator's initial and final states are as expected.

Solution:

The test pattern for this type of behavior is only applied when an action is expected to be executed when a specific condition — or set of conditions — dependent on readings collected by one or more sensors is met. It consists of the following steps:

(1) Generate readings capable of triggering the expected action(s);
    This can be attained either by using physical sensors and possibly actuators, to produce readings within certain thresholds. This would involve:
    (a) Setup the sensor(s);
        This may include specifying the type of readings to be performed.
    (b) Setup the actuator(s);
        This may include resetting the actuator to a particular state.
    (c) Setup a gateway (optional);
        This may include configuring it to maintain a log of the data collected, so that it can be compared with the data persisted.
    Alternatively, this could be achieved without using actual sensors, but by injecting data directly to the IoT system.
(2) Verify if the readings persisted within the IoT system are capable of triggering the expected action(s);
(3) Verify if the action was performed, within the pre-set maximum delay.
    It is important to account for possible delays between the triggering of the action and the actual execution. Verifying if the action was executed may be achieved by checking the state of an actuator or the readings performed by a sensor.

Example:

Take once more an IoT system which includes temperature and air quality sensors, which are setup to periodically perform measurements of the environment the system is in. Consider now that actuators are installed to control the opening and closing of window blinds given the average ambient temperature. Using this pattern it becomes possible to ensure that this pre-set action is triggered and executed, by providing a set of inputs which including the test runtime, sensor settings — with regard to id and type of readings performed —, the means for accessing the data collected, and the details about action itself — with respect to the triggering condition(s) and the expected final states.

3.5 Test Actuators

Name:

Test Actuators

Context:

Although IoT devices can be capable of contextual awareness, sensing capability, and some degree of autonomy, several IoT applications require the execution of actions upon command.

Problem:

It must be ensured that an actuator is capable of executing actions upon command, changing its state accordingly, under normal operation.

Therefore, it is necessary to:

- Check that the expected action has been triggered;
- Check that the action was performed with a maximum delay, ensuring that the actuator's initial and final states are as expected.

Solution:

The test pattern for this type of behavior is only applied when an action is expected to be executed. It consists of the following steps:

(1) Setup the actuator(s);
    This may include resetting the actuator to a particular state.
(2) Execute one or more commands;
(3) Verify if the actions were performed.
    It is important to account for possible delays between the triggering of the action and the actual execution.

Example:

Take once more an IoT system which includes temperature and air quality sensors — setup to periodically perform measurements of the environment the system is in — and actuators that control the opening and closing of window blinds given the average ambient temperature. Consider now that it should be possible to issue commands that open or close those blinds. Using this pattern it becomes possible to ensure that those commands executed, by providing a set of inputs which include the test runtime, actuator settings — with regard to id and communication details — and a set of commands and expected states.

4 DISCUSSION

The identification and documentation of these test patterns will enable the adoption of standard testing methods, allowing developers and test experts to test IoT solutions following similar strategies to address similar needs, ultimately leading to an improve in the quality of IoT solutions.

The approach described enhances reusability, as the test patterns can be applied to various scenarios to test recurring behaviours in the scope of IoT. Furthermore, this approach can be extended. More in-depth research can be carried out to, on the one hand, reveal additional test patterns in the scope of IoT-based systems, and, on another hand, document the new and pre-existing patterns following pattern template documentation standards.

It is also possible to develop test support tools — like *Izinto* [10] — that implement these patterns, further decreasing the effort put into testing IoT systems.

5 CONCLUSIONS AND FUTURE WORK

This paper presents five IoT Test Patterns to test IoT applications: Test Periodic Readings, Test Triggered Readings, Test Alerts, Test Actions, and Test Actuators. This pattern-based approach to IoT testing promotes reuse, as the test patterns put forward can be applied to various scenarios to test recurring behaviours in the scope of IoT.

Future work may include the definition of additional test patterns for IoT ecosystems for instance, covering aspects such as connectivity, security, scalability, and performance, along with other non-functional requirements. This could be achieved, for instance, by applying Machine Learning techniques to recognize a set of recurring behaviors of IoT applications to which can correspond a specific test strategy. Another possibility is to create test patterns that match existing design patterns within the scope of IoT.

REFERENCES

[1] Gullena Satish Chandra. 2016. Pattern language for IoT applications. In *Pattern Languages of Programs Conference*.

[2] Pedro Costa, Ana C R Paiva, and Miguel Nabuco. 2014. Pattern based GUI testing for mobile applications. In *Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014*. 66–74. https://doi.org/10.1109/QUATIC.2014.16

[3] Joao Pascoal Faria, Bruno Lima, Tiago Boldt Sousa, and Angelo Martins. 2013. A testing and certification methodology for an Ambient-Assisted Living ecosystem. In *2013 IEEE 15th International Conference on e-Health Networking, Applications and Services, Healthcom 2013*. 585–589. https://doi.org/10.1109/HealthCom.2013.6720744

[4] Gerard Meszaros and Jim Doble. 1997. Pattern Languages of Program Design 3. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Chapter A Pattern Language for Pattern Writing, 529–574. http://dl.acm.org/citation.cfm?id=273448.273487

[5] Rodrigo M L M Moreira, Ana C R Paiva, and Atif Memon. 2013. A pattern-based approach for GUI modeling and testing. In *2013 IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013*. 288–297. https://doi.org/10.1109/ISSRE.2013.6698881

[6] Ines Coimbra Morgado and Ana C R Paiva. 2015. Test Patterns for Android Mobile Applications. *Proceedings of the 20th European Conference on Pattern Languages of Programs* (2015), 1–7. https://doi.org/10.1145/0000000.0000000

[7] Amy Nordrum. 2016. Popular internet of things forecast of 50 billion devices by 2020 is outdated. https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[8] D Pavithra and R Balakrishnan. 2015. IoT based monitoring and control system for home automation. In *2015 Global Conference on Communication Technologies (GCCT)*. 169–173. https://doi.org/10.1109/GCCT.2015.7342646

[9] Anneke Pehmöller, Frank Salger Capgemini, and Stefan Wagner. 2010. Patterns for testing in global software development. (10 2010).

[10] Pedro Martins Pontes, Joao Pascoal Faria, and Bruno Lima. 2018. *Izinto*: A Pattern-Based IoT Testing Framework. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*.

[11] António Ramadas, Gil Domingues, João Pedro Dias, Ademar Aguiar, and Hugo Sereno Ferreira. 2017. Patterns for Things that Fail. In *24th Conference on Pattern Languages of Programs (PLoP 2017)*.

[12] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2016. Internet of things patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs - EuroPlop '16*. 1–21. https://doi.org/10.1145/3011784.3011789

[13] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2017. Internet of Things Patterns for Devices. In *Proceedings of the 9th International Conferences on Pervasive Patterns and Applications - PATTERNS 2017*. 117–126.

[14] Pallavi Sethi and Smruti R. Sarangi. 2017. Internet of Things: Architectures, Protocols, and Applications. https://doi.org/10.1155/2017/9324035

[15] Sidra Siddiqui and Tamim Ahmed Khan. 2017. On Test Patterns for Cloud Applications. In *Proceedings - 14th International Conference on Frontiers of Information Technology, FIT 2016*. 57–62. https://doi.org/10.1109/FIT.2016.019